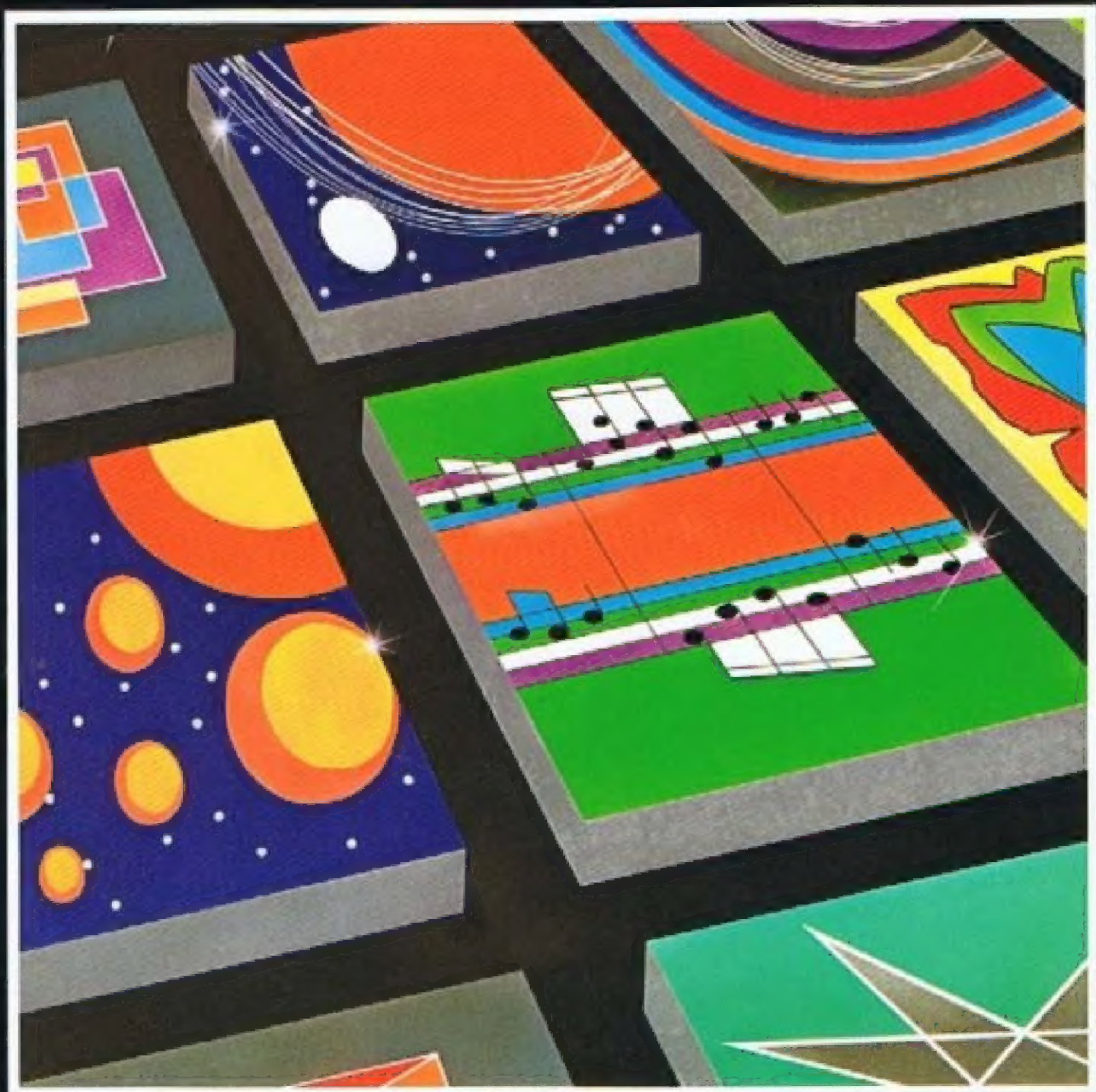


INCLUDING  
**SIMON'S BASIC**

# commodore 64 **Sight & Sound**



**JOHN J. ANDERSON**



# Commodore 64 Sight & Sound



John J. Anderson



Creative Computing Press  
Morris Plains, New Jersey



## Acknowledgements

I would like to extend the most sincere thanks to the following people for their gracious help with this book: Peter Kelley, for his fine illustrations and unfailing friendship; Neil Harris, of Commodore, for technical assistance and occasional morale boosting; Robert Alonso, for his expert programming assistance and reliable good humor; Andrew Simon, for having the insight to create the tool we've all been waiting for; Paul Farrell, for his faith and ever-energizing hysteria; and most of all to Laura Conboy for the enthusiasm to see this project through.

— John J. Anderson

The Author and the Publisher have made every effort to verify the accuracy of the information contained in this book. However, neither the Author nor the Publisher assumes any responsibility for the use of this information, nor for any infringements of patents or other rights of third parties that may arise from the use of the information in this book. Neither the Author nor the Publisher assumes any liability for any damages that may result from the information contained herein.

Commodore and Simon's Basic are registered trademarks of Commodore Business Machines, Inc. Apple and Apple Macintosh are registered trademarks of Apple Computer, Inc.

### Commodore 64 Sight & Sound

Copyright© 1984 Creative Computing Press

All rights reserved. No portion of this book may be reproduced—mechanically, electronically, or by any other means, including photocopying—without express written permission from the publisher.

### Library of Congress Cataloging in Publication Data

Anderson, John J.

Commodore 64 sight & sound.

1. Commodore 64 (Computer) — Programming. 2. Basic (Computer program language) 3. Computer graphics. 4. Computer sound processing.

I. Title. II. Title: Commodore 64 sight & sound.

QA76.8.C64A63 1984 001.64'2 84-12178

ISBN 0-916688-58-5

Creative Computing Press  
39 East Hanover Avenue  
Morris Plains, New Jersey 07950 USA

### Manufactured in the United States of America

86 85 84 987654321

Edited by Laura Conboy  
Cover design by Susan Gendzwil  
Cartoon illustrations by Peter Kelley  
Macintosh-generated diagrams by John Anderson



For Lauren, without whom I would most assuredly turn into a quivering lump of guacamole.



# Contents

<b>Introduction</b> .....	ix
<b>Chapter 1 Setting Up</b> .....	1
Powering Up .....	1
Getting the Picture .....	3
Tuning It In	
Seeking Resolution	
<b>Chapter 2 What a Difference a Drive Makes</b> .....	9
Shifting Into Drive .....	10
Formatting a Disk	
Saving Your Program to Disk	
Verifying Your Program	
Loading Your Program From Disk	
Obtaining a Directory	
Deleting a File	
Write-Protecting a Disk	
MENU: Or, the Easy Way .....	15
Driving the Wedge .....	19
<b>Chapter 3 The Key to the Commodore Keyboard</b> .....	21
The Caps Mode .....	21
The Quote Mode .....	22
Other Things to Know About the Keyboard	
<b>Chapter 4 Basic Graphics and Sound</b> .....	25
Basic Basics .....	25
Deferred & Direct Commands	
Printing From Basic	
Coloring Things In .....	32
Changing Colors With POKE	
Looping in Color	
IF/THEN	
FOR/NEXT	
Color Graphics From Print Statements	
The REVERSE Mode	
Making Loops Fruitful	
Easy Animated Graphics .....	45
Shift-Inverse Heart	
Inverse Bracket	
Saving Low-Res Screens	
Sound From Basic .....	52
<b>Chapter 5 Getting Into Simon's Basic</b> .....	57
Simon's Basic Commands .....	57
COLOUR	
KEY	
DISPLAY	
AUTO	
RENUMBER	
PAUSE	
ON ERROR:GOTO	
OUT	
Simon Meets Low-Res Graphics .....	66
SCRV and SCRLD	
BCKGNDS	



FLASH and OFF	
BFLASH	
FILL	
MOVE	
SCROLL	
Hi-Res Graphics .....	74
HIRES	
LINE	
REC	
A Closer Look at Plot Types	
Multi-Res—The Best of Both Worlds .....	80
MULTI	
Plot Types in Multi-Res	
LOW COL	
HI COL	
PLOT	
TEST	
CIRCLE	
ARC	
ANGLE	
PAINT	
BLOCK	
DRAW	
ROT	
Displaying Text in Hi-Res and Multi-Res .....	100
CHAR	
TEXT	
NRM	
CSET	
<b>Chapter 6 MOBbing Up in Simon's Basic .....</b>	<b>105</b>
Sprites Are Mobs .....	105
DESIGN	
@	
CMOB	
MOB SET	
Mobs in Motion .....	113
MMOB	
RLOCMOB	
MOB OFF	
DETECT and CHECK	
<b>Chapter 7 Sound From Simon's Basic .....</b>	<b>125</b>
Simon's Sound Commands .....	125
VOL	
WAVE	
Noise Waveform	
Square Waveform	
Sawtooth Waveform	
Triangular Waveform	
ENVELOPE	
MUSIC	
PLAY	
<b>Appendix .....</b>	<b>135</b>





# INTRODUCTION

---

Well you finally went and did it. You waited and waited, biding your time until a machine with loads of capability appeared at a reasonable price. You weighed your options carefully. You narrowed your choice down to three machines. Then two. You wavered. You reconsidered. And then you went out and bought yourself a Commodore 64 microcomputer.

Take a moment to pat yourself on the back. You've made an excellent decision. The Commodore 64 is just about the most computer you can buy for the money. Its graphics capability is unparalleled for versatility and sophistication, its sound capability rivals that of dedicated sound synthesizers, and its 64K of RAM (Random Access Memory) is more than ample for nearly any application you can imagine. It's difficult to believe that all of this power, all of this amazing technology, could be sitting right in front of you in such a compact, inexpensive package.

And there's more—much more. Lots of software is available for your machine, and you'll find that many of these packages are of very high quality and even constitute a new generation of personal computer software. Familiarize yourself with what's available so that you can make educated decisions on what packages you should purchase. At the end of this book, there is a short appendix summarizing a few of the more popular graphics and sound programs available for your machine as well as a listing of the hardware peripherals that you can use to make sound and graphics easier and fun.

But don't stop there. Go beyond the limits of the software and hardware and learn a bit about how your machine works. You may have already taken a stab at this only to quickly discover that your machine is lacking in documentation. Maybe you tried to program graphics and sound from Basic but just got frustrated at its complexity. Don't give up. Many intelligent people just like you have come up against the same wall. The Commodore 64 is capable of sophisticated sprite graphics and sound effects, but when using plain old Basic, programming this power becomes a nightmare.



Here is an example of a simple animation with one sound effect written in C-64 Basic:

---

```

1 REM PROGRAM 0
2 REM A TYPICAL BASIC SPRITE ANIMATION
3 REM PLEASE *DON'T* TYPE THIS ONE IN!
4 REM-----
20 POKE52,48:POKE56,48:CLR:V=53248:POKEV
+22,0:POKEV+17,16:J=1:U=54272:W=128
25 DEFFNA(L)=(LAND15)+192:POKEU+4,16:POK
EU+5,2:POKEU+6,0
30 POKEV,10:POKEV+1,204:POKEV+39,0:R=V+1
6:POKEV+37,7:POKEV+38,4:POKEU+24,15
40 POKE2040,194:POKEV+23,1:POKEV+29,1:PO
KEV+21,1:POKEV+28,1
45 FORM=0T0255STEP2:POKEV,M:POKE2040,FNA
(M/2):GOSUB90:NEXT:POKEV,0:POKER,1
50 FORM=0T096STEP2:POKEV,M:POKE2040,FNA(
M/2):GOSUB90:NEXT:POKEV,0:POKER,0
60 J=1-J:POKEV+27,J:GOTO45
90 Q=M/32:IFQ=INT(Q) THENPOKEU+1,50:POKEU
+4,W+1
91 IFQ-INT(Q)=.5 THENPOKEU+1,250:POKEU+4,
W+1
92 IFABS(Q-INT(Q)-.5)=.25 THENPOKEU+4,W
100 RETURN

```

---

The problem with the other Commodore 64 sight and sound books out on the market is that they seriously try to explain programs like the one shown here, leaving the reader feeling confused, intimidated, and unable to make heads or tails out of the material presented. Book after book, the user gets no closer to graphics and sound mastery—the only cumulative effect is the sinking feeling that these complicated concepts will never be understood. It's depressing to think about how many adults and children have been turned off to the C-64 and probably to computers in general as a result of this needless frustration.

This book is designed to break through the walls that C-64 users so often come up against when trying to understand sound and graphics. The major tool we'll be using to break through these walls is *Simon's Basic*, a new Commodore version of Basic that makes programming much easier. It was designed in England by Andrew Simon, who was appalled by how difficult plain old Basic was to learn and use. He realized that for a language to work, it had to have an easily understood vocabulary—one that allows you to express what you need to say in a simple and clear way.

So, Mr. Simon went ahead and designed a new and powerful Basic, giving it his name. And by the time he was finished he was 16 years old.

The result of Simon's efforts is a package which is one of the best aids you can buy to help you learn how to program graphics and sound on the Commodore 64. It comes as a cartridge and fits into the slot on the back of your machine. For a reasonable price, Simon's Basic supplies you with a powerful new Basic that includes special graphics characters and sound



commands. Using Simon's Basic, you can finally get your hands onto the real power of your machine.

If you don't have Simon's Basic, you are really limiting yourself as to what you will be able to do with your machine. Not only is the documentation accompanying the machine extremely lacking, but the Basic itself, on ROM (Read Only Memory) inside the C-64, works against the novice. As you can see from our sample program, graphics and sound commands from plain old Basic require extensive machine language POKes to get anything done. Because they look so much alike and divulge no clue as to the operation they provide, these commands are impossible to understand without extensive study.

So it isn't you, the user, who is inferior for not understanding graphics and sound from plain old Basic. It is the language and documentation that are inferior. Be aware, however, that the documentation accompanying Simon's Basic is as shaky as anything we've seen from Commodore. But the language itself is a vast improvement. Using this book, you will quickly understand how Simon's Basic works. You will master its graphics and sound commands. You will write your own programs to create color, sound effects, and music. And all you will need to accomplish these wonderful goals is Simon's Basic and this book.

Let's get going. First we'll go over some other topics like setting up your system, making sure the picture you are getting is the very best one possible, and learning how to use the disk drive. We'll also look at the keyboard of your C-64 and unlock its secrets.

Once you've come that far, you will be ready to start programming graphics and sound. We'll begin from plain old Basic, doing some simple graphics and learning some simple commands. We'll also look at some examples of sound programming from Basic.

Then we shall embark on our exploration of Simon's Basic. We will work through commands for low-res and hi-res graphics and for sound control. Examples of each command are provided, and a group of programs is presented in each chapter to serve as a springboard for your own original work.

So boot up, and let's get started!



# Commodore 64 Sight & Sound





## SETTING UP

---

Setting up your C-64 is very simple. You've got a power supply box and line, a TV or monitor cable, and a cassette or disk drive cable to plug in. The jacks on the ends of these plugs are designed so that they cannot plug into the wrong place. The illustration included in this section shows a typical set-up with a TV and a disk drive system.

If you're using a cassette drive instead of a disk drive, the jack plugs into the socket on the top lefthand side of the back of the computer. It will only plug in where it belongs, and you cannot plug it in upside down—the jack won't let you.

If you have a dedicated monitor rather than a TV, you will require a monitor cable. It plugs in next to the disk drive cable on the back of the C-64. The other side goes to separate audio and video plugs on the monitor. Make sure your monitor cable is designed for the C-64—other cables may work but will provide an inferior picture. For information on wiring your own monitor cable, see the upcoming section on *Getting the Picture*.

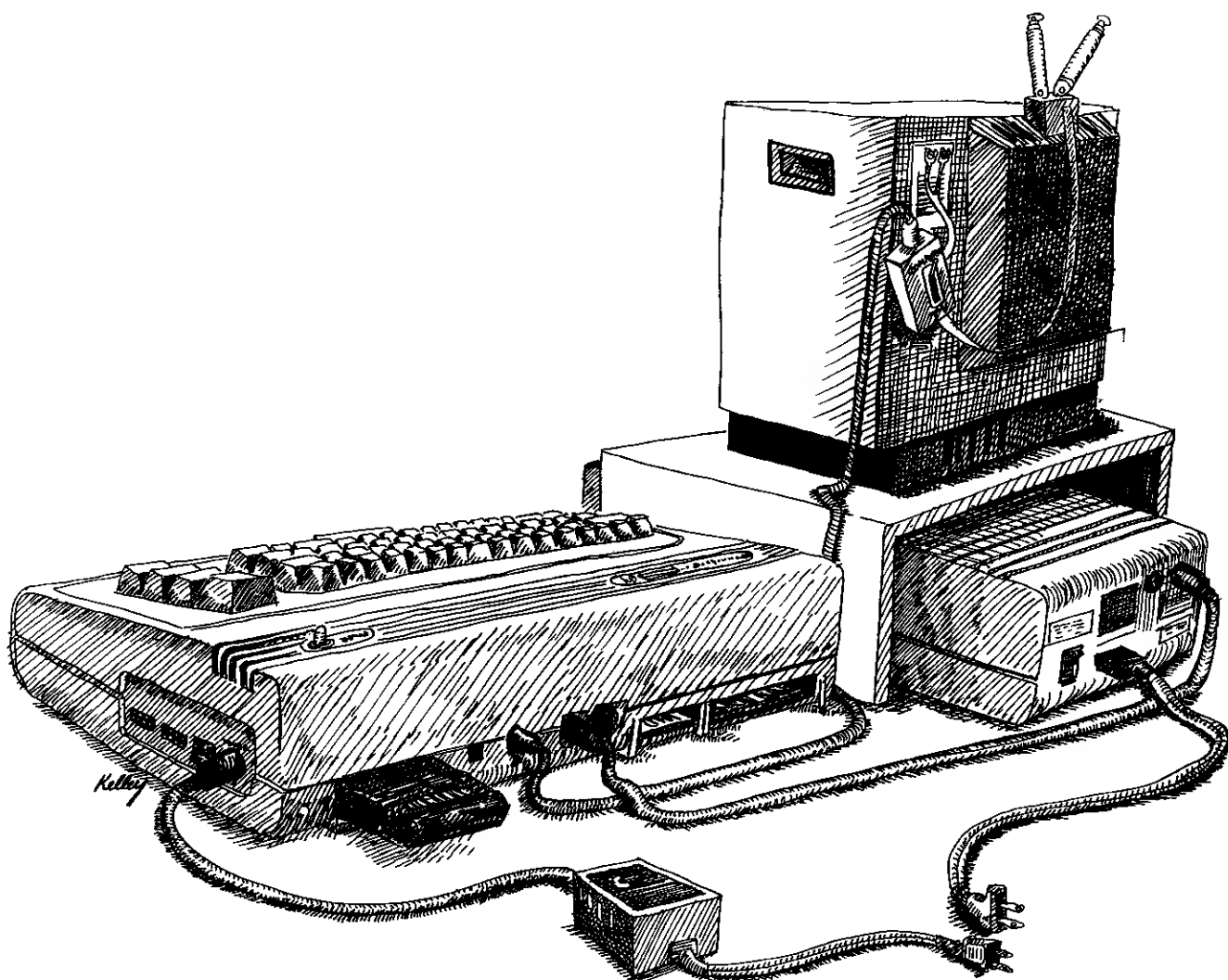
Your disk drive has a built-in power supply. You will attach the drive to its own power cable. You will also attach the drive to the other side of the disk drive data cable which attaches to the C-64.

The far side of the TV cable attaches to the plug on a silver switch box (you may already be familiar with this piece of hardware). The terminals on the switch box are then attached directly to the VHF (that's VHF, not UHF) input on the back of the set. Attach the antenna input of the TV to the side of the switch box. Then you can select the TV or computer signal by sliding the switch on the switch box back and forth.

That's about all there is to setting up. Now you're ready to start switching things on.

### POWERING UP

Always turn on the power to the disk drive before you turn on power to the computer; otherwise the computer may not know that it has a disk drive connected to it. Make sure that the TV is on and tuned to channel 2 or 3. If you have a TV channel 2 in your area, turn to channel 3. If there is a channel 3 in your area, tune to channel 2. Then set the toggle switch (located on the back of your C-64 between the TV output and the ROM cartridge slot) either left or right—left for channel 2, and right for channel 3.



If you have Simon's Basic, plug it in the ROM slot (*before* you turn on the computer) with the label side up. *Never* plug a ROM cart in or out of the computer while the computer is on.

All set? Okay, turn on the computer. Use the TV tuner to bring in the picture. (Tips on getting the best picture come in the next section.) If you are getting no picture at all, make sure that you are cabled properly, that the switch box is set to "computer," and that the channel toggle on the back of the 64 is set correctly.

If you are getting a picture but not a very good one, try adjacent channels on the TV. You may be set up for whichever channel you are not currently on.

When you turn the computer on ("power up"), the red light on the disk drive should come on for a second or so, then go off again. If this does not happen, make sure your disk drive is on. If it is not, turn the computer off ("power down"), turn the drive on, then turn the computer back on. Now the red light should blink at you.

The green power light on the disk drive is very hard to see at times. Use a little extra care to make sure you turn the disk drive off at the end of a session with the computer. The drive gets very hot when left on for long periods, and this can impair performance.

When the C-64 comes on, the red light on its top panel will blink on. If this doesn't happen, check that the power supply is plugged into the wall and that it is also correctly plugged into the Commodore 64. If it is, you may have a problem that requires service on your C-64 or power supply (or you're having a blackout).

That should do it. You're getting a picture on the screen, and all the components of your system are ready to go.

And remember, the side of your C-64 that has the keys on it always faces up.

■■■■■■■■■■

## GETTING THE PICTURE

Before we go on, take a good look at the video output on your C-64 system. Does it look okay? You will need to look at many program listings, and clearly see strange shapes (like special graphics characters), so a good screen output is an absolute necessity. If you can't clearly see what you are doing with the C-64, you simply won't be able to enjoy this book.

If you've set up and tested your C-64 as indicated in the previous section, and the picture looks terrific, don't read another word here. Skip right on ahead to the chapter on using the disk drive.

If you're still reading, then you are not happy with the picture you are getting. Don't despair. There are a few good ideas to try which can improve your reception.

At under \$500 for a C-64 and disk drive, the Commodore is a hard system to beat. It offers sprite graphics, built-in Basic, exceedingly good three-voice sound synthesis, and some incredible peripherals: graphics tablets, light pens, piano keyboards, and drum synthesizers. Add to this the C-64's advanced gaming capabilities, and you've got the perfect home computer, right?

Well, maybe. The fact is that the reputation of the 64 has been literally clouded by reservations concerning the quality of its video. Some potential C-64 customers have claimed that the pictures they have seen on screens at their local computer store were singularly unimpressive. Others wonder whether early models caused interference on their screen.

Scout's honor and for the record, the Commodore 64—even the early machines—can output an acceptable color video signal. In fact, this book was written using one of the first machines to come off the assembly line. Due to a little care in set-up, its video quality is just fine.

Before you pass judgment on the reception of a unit in a department store, computer store, or even your own home, you should understand some of the obstacles to good video, and the requisite steps to overcome such problems. Commodore 64 video is definitely among the more persnickety signals you are bound to come across, and without a doubt it requires some special treatment. You must realize that you can't get the same picture possible on a NEC monitor using the 20-year old Philco in your den. It is true, however, that this is a fact a salesperson is loath to admit just prior to a sale.

By the same token, only very rarely is a video signal well-tuned in a

retail store. Think of how lousy the pictures look on the ordinary TVs in a typical department store, then think of the additional skills required to set up the C-64. You can always do better on this score at home. There are limits however as to how well you can do with an ordinary TV.

The most dramatic way to improve the output of a Commodore 64 is through the purchase of a dedicated video monitor. Here's why.

When you hook a computer to your home TV set, you route the signal through a special modulator. Then the video and audio signals are sent through this modulator and into the antenna input and tuner of the set, just as if it were this week's segment of your favorite TV show. While this is certainly a convenient means of accessing a pre-existing CRT, it has distinct limitations. The signal is necessarily weakened in passing through the tuner section of the set. If only the video and audio signals could bypass all of that and go directly into the circuitry designed to get the picture on the screen and the sound into the speaker, then the quality of the signal would be dramatically improved.

That is exactly what inexpensive home monitors are designed to do. Typically such monitors cost about \$250 and have built-in audio amplifiers with speakers to handle sound.

The difference between the video quality of the 64 on a regular TV and one of these monitors is very striking. The monitor video looks twice as sharp, and can take much greater color saturation without "bleeding," which is the extremely unfortunate smearing of borders between clashing colors. Bleeding seems to be the foremost complaint of Commodore 64 owners.

If you've been thinking about getting a monitor, you should be aware of another potential boon to be associated with it: no longer will any other member of the family be denied the TV while you are playing with the computer.

Whether you have a monitor or not, there are further steps you can take to make sure the signal coming out of the 64 results in the best possible picture. One of the first points one realizes about tuning in the video is that the color must be set quite differently from the levels we have come to think of as "normal" as a result of the time we've logged with Apples and Ataris. In fact, many monitors have "detents" (preset notches) for default settings which always seemed about right for those machines. Not so with the 64.

■■■■■■■■

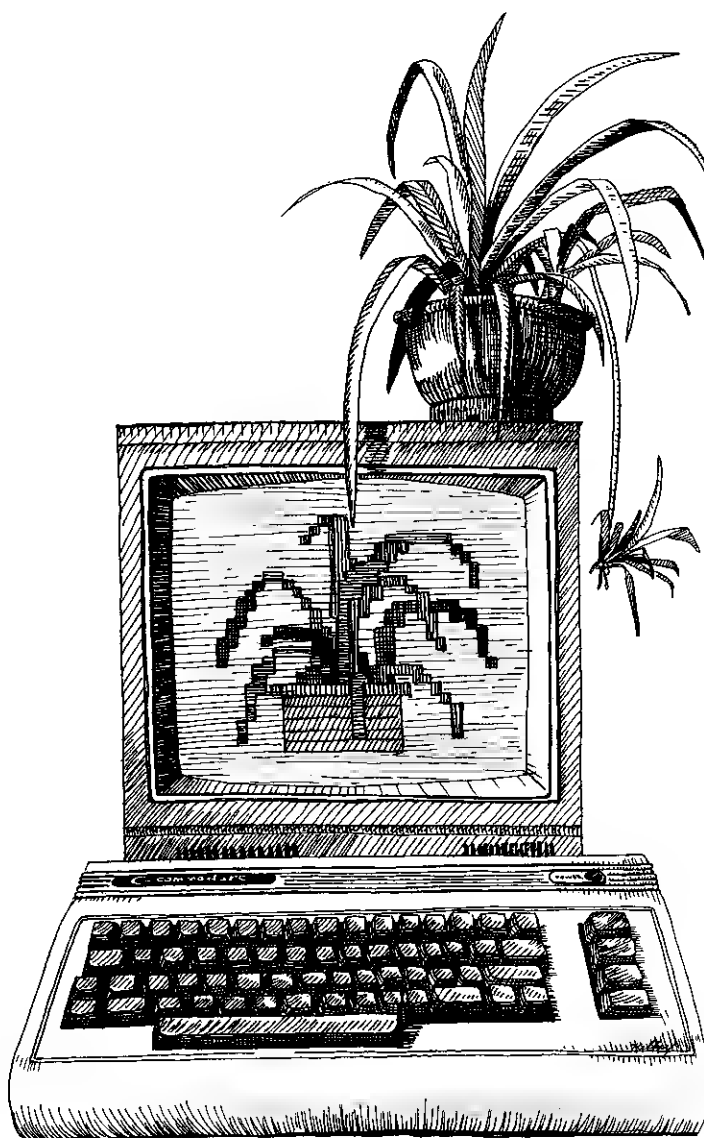
## Tuning It In

The most important change is to back off on the color level. In almost every case, this will immediately improve reception. The Commodore chroma level (level of color saturation) is designed for less saturation than other machines as well as less than broadcast TV.

Next, *increase* the brightness and *decrease* the contrast of the picture. This, combined with lessened color intensity, will improve video quality substantially.

While colors on the 64 will be a bit less vibrant at these settings, they can still be very colorful. If they are looking washed out, go ahead and boost color saturation. The thing to look for is an overall balance between lesser than usual color and contrast, and greater than usual brightness.

Even these steps are not enough to thwart all cases of color bleeding, or some other peculiar problems of Commodore video. To understand more about these problems, we have to take a closer look at video technology (or raster technology) in general.



## Seeking Resolution

In the design of any machine there are trade-offs. In the design of color computers, high-end designers may commit to what is called RGB (Red Green Blue) technology, using special, expensive monitors to create impressive resolution and clarity. On some of these systems nearly photographic results can be achieved, and color bleeding occurs only when desired, by making borders ragged or blended. Modulation to a conventional color TV is automatically ruled out on these systems.

When surrendering to the restrictions of raster technology so that a signal can be pumped to a raster monitor or conventional television, its limitations are passed to the computer designer. The resolution of such systems is more limited, and some kinds of color bleeding are avoidable only by staying away from the certain color combinations that cause them. These limitations are defined by the system, and nothing inside the computer can get around them.



Let's compare. On the Apple II computer, character sets (predefined sets of letters and numbers) are always white on a black background, no questions asked. On the Atari, backgrounds can be any of 256 colors, but text must always be white or black, at least from plain vanilla Atari Basic. In many cases, however, a lack of contrast or severe color bleeding will make certain color combinations unacceptable.

On the Commodore 64, there is much greater flexibility. Character sets in any of 16 colors can appear on a background of any of those same 16 colors. Not only can the color of multiple characters be controlled individually, but in a very straightforward manner, with direct keyboard commands (pressing CTRL-WHITE turns the cursor, and all subsequent characters, white). Combined with the keyboard graphics characters, this flexibility offers a powerful graphics tool.

It also creates problems. Some color combinations render characters utterly indistinguishable. Then there are the default color settings, which are automatically provided when you power-up. They lack contrast and are difficult to work with for long. Though there is no bleeding, the light blue characters on a dark blue field with a light blue border can be eye-strainers. (This problem is solved automatically by Simon's Basic and on the portable SX-64, both of which use a white background.)

No wonder some people complain about video on the 64, with such a case of the blues. At the very least, many a C-64 user will press CTRL-WHT to turn the cursor and characters white on the default (blue) background. That is a real help.

Owners of new machines will probably wonder why an entire section of this book is devoted to video. The latest machines have a very clear and strong video signal, with very little bleeding. Owners of early C-64s may be surprised to see how good the new machines look.

But the older machines have some problems, despite all the home remedies we shall list ahead. Some color combinations used in this book may cause bleeding problems on older machines or accentuate light parallel lines running vertically down the screen.

Though all the programs in this book have been created on a vintage C-64 (pre-rainbow logo), we cannot guarantee that every program will look terrific on every early-model 64. Sorry, folks.

In addition, many disappointed owners of older C-64s, some with color monitors, have found that their screen outputs are suffering from a problem known as "the sparkles." This ROM problem results in a distracting glitter effect across the screen. One of the 64s we have here at the lab has got a really bad case of the sparkles. It renders many programs very hard to watch, and certainly spoils graphic effects. Even text screens are affected. Unfortunately, there is nothing you can do about the sparkles on your own. You'll have to send your C-64 in for service to correct the problem.

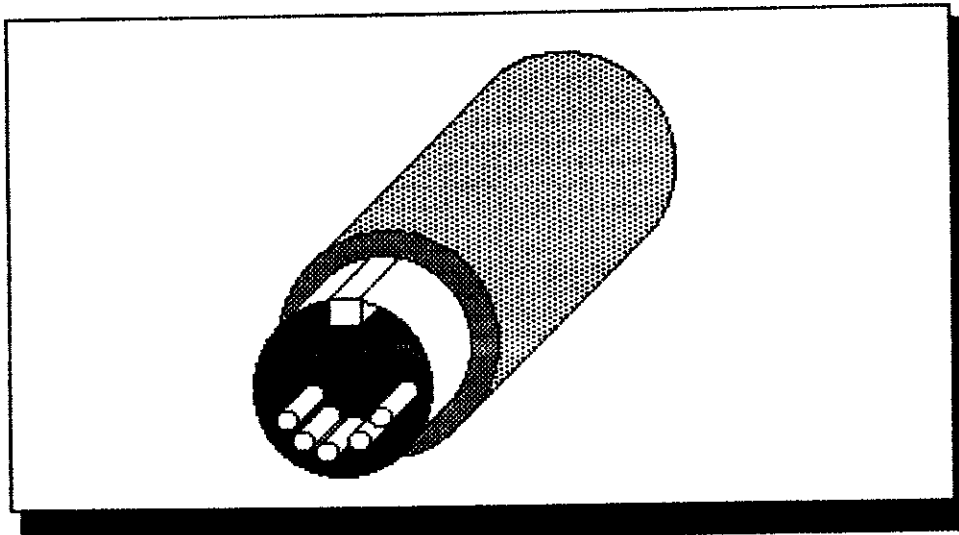
If you do have the good fortune to own a color monitor, one thing you should be sure about is the monitor cable you use. Many of the commercially available cables which do wonders for the Atari and Texas Instruments computers are not capable of providing maximum video quality for the C-64.

If you are buying a monitor cable, make sure it is specifically designed for the Commodore 64. Better yet, consider saving the money on a pre-assembled cable, and make the cable yourself.

What you'll need is a five-pin DIN male connector, an RCA male phono

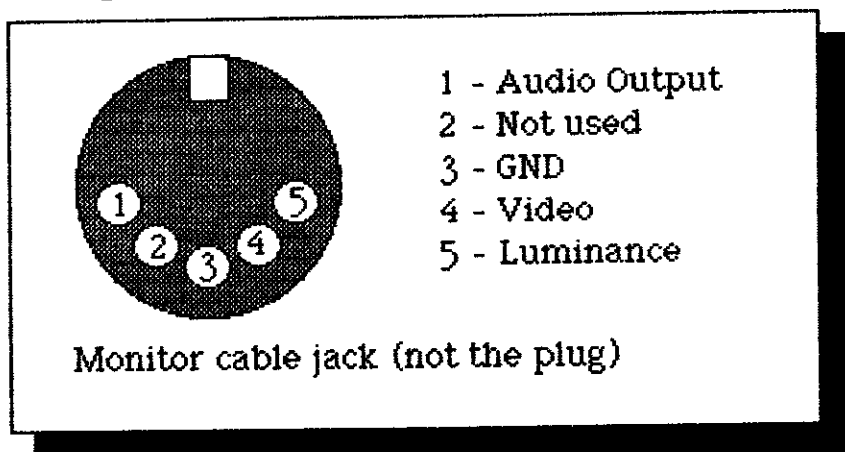
plug, and a male mini-plug. When you buy a DIN connector, you must make sure that the pins look like this:

Finding the right five-pin DIN plug.



There are several kinds of five-pin DINs, and only this kind will fit into the monitor jack on the back of the C-64. The wiring diagram for a monitor cable is as follows:

Wiring a monitor cable.



If you're too young or don't know how to handle a soldering iron, buy a cable from your local computer store. This project is only for folks who have had a little soldering practice.

If you are still reading and are still dissatisfied with the video quality of the 64, here are some fixes you can try:

- You can get a heavy-duty shielded TV cable, with gold contacts, for about \$10. This replaces the cheaper, thin TV cable that comes with the C-64. The heavy shielding insulates the video signal from interference and static and will provide about as good a signal as you can get to the TV switch box. Ask your audio, video, or computer dealer for a heavy-duty gold-contact cable with a male RCA phono plug on each end.
- You can bypass the switchbox entirely. Sometimes the culprit in video problems is that innocent-looking little silver box itself. Cut one of the plugs off the video cable and strip the leads. Attach these leads directly to the VHF antenna terminals of the TV. It may be annoying to connect and disconnect these leads when going back and forth from regular TV, but it might be worth the extra effort.
- You can route the signal through a home VCR. Home video tape recorders have their own RF modulator, and chances are it has a better RF modulator than does your C-64. Wire a monitor cable as shown above, with RCA male phono plugs for both audio and video input. Plug the cable into the video and audio-in jacks on your VCR, and see if your video quality isn't substantially improved.
- You can adjust the TV controls. Keep experimenting with the TV control settings until you get the best setting for the C-64. As we said earlier, these settings may be very different from those your TV uses normally. Again, it will be a hassle to have to adjust everything when going from TV to the computer, then back again. But it will be worth it for an improved picture.
- You can invest in a monitor. This is the most dramatic way to improve your video quality.



## WHAT A DIFFERENCE A DRIVE MAKES

A computer isn't worth very much if it can't store and retrieve programs. If your computer couldn't do these things, it would forget a program as soon as it was turned off, and you'd have to rekey your entire program when you turned the computer back on.

Clearly it is preferable to save a program on some sort of medium so that you can simply load the program whenever you want to run it. Though cassettes work simply and perfectly well for this purpose, floppy disks and a disk drive turn the C-64 into a *real* computer.

Disk drives are *random access* devices. This means that a disk drive can easily go back and forth across a disk, accessing any program or data without needing to fast-forward or rewind. One clear advantage of using floppy disks instead of cassette tapes is the ability to go right to what you are looking for, without having to pass over all that comes before.

A single disk can hold a tremendous amount of data. Each Commodore disk can hold about 170K (approximately 170,000 characters), or nearly three times the total memory capacity of the C-64 itself. A C-60 cassette tape can hold a lot, but accessing through that much material would take an eternity—not to mention the time used in having to flip sides of the tape.

The really good news is that Commodore disk drives are about the least expensive drives on the market. If you shop around, you can get one for under \$250, and even that price continues to shrink.

Working with a disk drive is a little trickier than working with a cassette drive, but after you catch on, you won't want to use cassettes ever again. Disks can hold lots more programs than cassettes. And because the disk is a random access medium, it will quickly locate your programs for you so you won't have to note counter numbers as you do when using a cassette player.

One real trouble you can get into with disks is forgetting to do some things that let the drive know what's happening. If you make this kind of mistake, you may not be able to load or save programs, and you may even lose programs. So use a little extra care until you have learned everything you need to know about using floppy disks. Treat your disks and your drive well, and they will do the same for you.

■■■■■■■■■■

## SHIFTING INTO DRIVE

When using a disk drive, the first thing to remember is to turn it on, before you turn on your C-64. If your C-64 is already on, turn it off, turn on the disk drive, then turn the computer on again. This tells the computer that the disk drive is connected and ready to use.

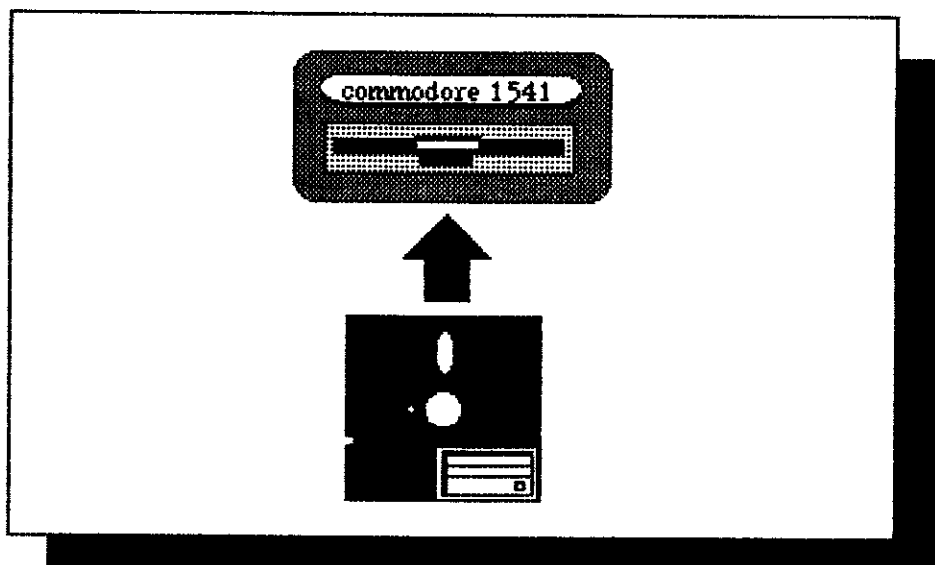
A good practice to get into is to never turn the disk drive on or off while there is a disk inside of it. This will help to avoid unnecessary disk damage. This is not really a problem with the newer 1541 drives but it is with the older 1540s—much data can be lost as the result of a drive turned off at the wrong time.

The drive is powered on when the green light on its front panel is glowing. Then, when the computer is turned on, the red light on the disk drive will come on for a few seconds. The computer now “knows” that it has a disk drive attached to it (i.e. the disk drive has been “initialized”).

The red light on the drive means that the disk drive is spinning. It acts like a traffic light. When it is red, don't insert or pull out a disk. Wait until the red light goes out before opening the drive door. To open the drive door, press on the black bar in the middle of the door. The door will snap open. It is a good idea to hold the bar with your thumb as you press it, so it doesn't snap open too hard.

Now you are ready to insert a disk. Hold the sleeve as shown in the diagram below:

Inserting a disk in the disk drive.



Make sure that the label side faces up and that your fingers don't touch the shiny disk surface through any of the windows on the sleeve. Slide the disk through the door of the drive, making sure that the oval window is the first part of the disk to go in.

Press the disk in all the way until it snaps into place. Then push down on the black bar to shut the drive door.

Now we're ready to control the drive from the C-64. The commands we'll use are a bit involved, and you may find them difficult to memorize. The first encounter with a Commodore disk drive can be a disheartening experience.

## Formatting a Disk

Fortunately, there is a better way to learn C-64 disk commands than having to memorize them. A special program called "Menu" will be introduced shortly. It will help you take care of many disk housekeeping chores. Before you allow yourself to become confused by cryptic command codes, bear in mind that you'll only have to type them once if you type in the Menu program. This will make the material which follows much easier to accept.

So, for the record, here's how to control the disk drive from Basic. Grin and bear it.

Before you use a blank disk to save your programs, the disk must be *formatted*. Formatting organizes the disk into "pages" that your C-64 can write to and read from. When you format a disk, you give the disk its own name and number. With a blank disk in the drive, the command to format it is issued in the following way:

```
OPEN 15,8,15,"NEWØ:DISK NAME,TWO-DIGIT NUMBER"  
RETURN
```

where:

disk name = your chosen filename

two-digit number = 00–99

Instead of calling the disk DISK NAME, you can name your disk whatever you like. You can even use multiple words with spaces between them. But remember that a filename, including spaces, can be no longer than 16 characters. Follow the name with a comma and then a two-digit number. For example, you might call your first disk DISK1,01. You would type

```
OPEN 15,8,15,"NEWØ:DISK1,01" RETURN
```

Don't forget the commas and to use Ø (zero), not O (letter O), after the NEW command.

Once you've done this, the red light will come on, and the disk will spin, guzzle, gurgle, and chirp for about two minutes. Don't fret over these strange noises. When the disk stops spinning, it is ready to use.

Always try to make the number of each new disk different from the ones that have come before. For example, you might call your second disk DISK2,02. If you give two disks the same name and number, you might confuse the disk drive when you switch from one to another.

After any and every format procedure, type

```
CLOSE 15 RETURN
```

to close the file you opened with the FORMAT command.

\*\*\*\*\*

## Saving Your Program to Disk

When you are ready to save a file to disk, follow these procedures:

1. Make sure that the disk drive was turned on first, and then the 64.
2. Make sure that the disk in the drive has been formatted, and that the program you want to save is in the memory of the C-64.
3. Type

```
SAVE "PROGRAM NAME",8 RETURN
```

where:

program name = your chosen filename

When using quotation marks, you can name your program whatever you wish. The number 8 tells your C-64 to save the program to disk. (If you leave off the comma and the 8, the C-64 will assume you are saving to cassette.)

4. When you press **RETURN**, the C-64 should tell you SAVING PROGRAM NAME and then READY. If it doesn't, repeat steps 1 through 3.

\*\*\*\*\*

## Verifying Your Program

It is not as important to doublecheck your program on disk as it is on tape. Just the same, you might want to make sure that the program is safely saved to disk, and that it exactly matches the program in memory. Type

```
VERIFY "PROGRAM NAME",8 RETURN
```

where:

program name = filename to be compared with program currently in memory

Remember that the 8 always tells the C-64 to use the disk drive.

The C-64 will now print SEARCHING FOR PROGRAM NAME, and then VERIFYING. If the program has been correctly saved, the C-64 will say OK.

If you see VERIFY ERROR, it means the program on disk does not match the program in memory. In that case you should try to save the program again. Usually the only time this will happen is when a disk is completely filled up. You'll probably encounter a filled disk only once in a long while, but it is still a good idea to keep a few blank formatted disks around. That way, when a disk is filled up, you will be able to save to a new disk without any problems.

\*\*\*\*\*

## Loading Your Program From Disk

To load a program, use this checklist:

1. Make sure that the disk drive was turned on first, *before* the C-64.
2. Make sure that the disk in the drive has been formatted, and that it contains the program you want to load.



## 3. Type

```
LOAD "PROGRAM NAME",8 RETURN
```

The name of the program you wish to retrieve goes in between the quotation marks. Again, the number 8 tells your C-64 to load the program from disk.

4. Press **RETURN**. The C-64 should now read out LOADING PROGRAM NAME and then READY when it is finished.

5. If something went wrong, you will see an error message. If it says ?FILE NOT FOUND ERROR you are either not typing the right name for your file or you do not have the right disk in the drive.

There are a few other commands that can come in very handy when you're using a disk drive, and you might want to get to know one or more of them.

■■■■■■■■■■

## Obtaining a Directory

What if you can't remember the name of the program you want to load? To see all the file names on a disk, type

```
LOAD "*",8 RETURN
```

Then type

```
LIST RETURN
```

This will display the name and number of your disk along with all the file names (programs) stored on it. If there are a lot of programs, the list may go by too fast to read. You can slow the list down by pressing CTRL, or stop the list completely by pressing RUN/STOP.

■■■■■■■■■■

## Deleting a File

What if you want to get rid of a program? Well, the first thing to do is make sure that you're working with the filename of a program that you really want to remove from disk. Next, type

```
OPEN 15,8,15,"SCRATCH0:PROGRAM NAME" RETURN
```

PROGRAM NAME will really be the name of the file you wish to delete. Remember to type a 0, not an O, after the word SCRATCH.

After the drive stops spinning, type

```
CLOSE 15 RETURN
```

to close the file you opened with the SCRATCH command.

One important note: you cannot use the SAVE command to update an existing file. If you try to save a program under a filename that already exists, you will not get an error message nor will the modified program have been stored.

Sometimes you will make revisions to a program and you'll want to save the new version under the same name as the old one, deleting the old file in the bargain. The way to replace an existing file is to type

```
SAVE "@0:PROGRAM NAME",8 RETURN
```

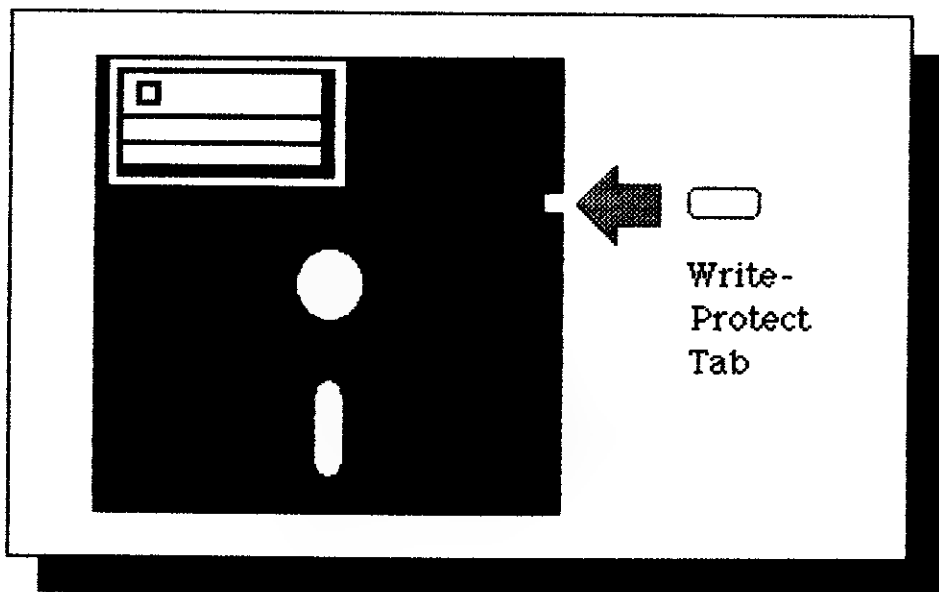
The @ key sits between the P and \* on the Commodore keyboard. Again, make sure the 0 is not an O.

\*\*\*\*\*

## Write-Protecting a Disk

Another good thing to know about disks is how to protect them from being mistakenly written to. Protecting a disk allows you to load programs which are on the disk but does not allow you to save new or revised programs. In effect, it keeps you from mistakenly destroying or changing programs that you want to keep. To protect a disk, all you have to do is cover the little notch on the side of the disk sleeve with a write-protect tab (these are the sticky silver tabs that come with your disks) or a small piece of tape.

Write-protecting a disk.



■■■■■■■■■■

## MENU: OR, THE EASY WAY

When a protected disk is in the disk drive, programs can be loaded from the disk, but no changes can be made to the disk. Any disk that you consider "finished" should be protected with a tab on that little notch. If you should change your mind and decide that you want to change what's on the disk, you can always simply pull off the tab or piece of tape.

Becoming familiar with the disk commands takes time and effort. Unfortunately, there are a lot of procedures to remember, and even something as simple as obtaining a disk directory must be done in several steps. Even more unfortunately, the manual that is included with the disk drive is difficult to decipher and quite intimidating.

As an alternative to memorizing all the command codes for each disk command, consider typing in and saving the following Menu program. This program simplifies understanding the commands we've just discussed. It also adds a few more disk management capabilities.

When you run the program, you are presented with a menu. You automatically receive a directory for the current disk in the drive. You may then pick any of a list of possible commands.

First the name and extender (the two-digit number which follows your filename) of the current disk are listed. Then the lengths, names, and types of files are listed. After the directory is completed, a menu prompt of choices appears. You pick a letter, hit **[RETURN]**, and the chosen process is automatically carried out for you.

Here is an explanation of all the functions Menu can perform:

- **(←) RUN** By pressing the left-arrow key, which is the top-left key on the C-64 keyboard, you can autorun any program in the directory. Hit **[RETURN]** to confirm your selection. Then the computer will ask for the filename of the program you wish to run. Enter it. Then press **[RETURN]** once more. The program you chose will load and run.

- **(F) FORMAT** Before you can store any information on a blank disk, it must first be formatted. Choose the format selection. The program will then give you a chance to insert the disk you wish to format. Spare yourself agony by making sure the disk you are about to format does not contain a file you cannot do without. Formatting destroys all existing files.

The program will then give you a chance to insert a blank disk and ask you for a disk name. After you have entered a disk name, you will be prompted to enter an extender. When you press **[RETURN]** again, the disk will format automatically. Again—this kills whatever may have been on the disk, so be careful about what you format.

- **(C) Copy** This function allows you to copy a file under a new filename. You will be prompted for a source and a new filename. An identical file will then be created under the new filename.

- **(E) Erase** This function deletes a file from disk. It prompts you for a filename and then gives you one chance to reconsider before deleting the file.

- **(D) Directory** This function lists the files on the disk. This will happen automatically once the program is run. It is a good idea to run a directory right after any file manipulation from Menu, to make sure the operation has come off successfully.

- **(\*) Check error status** This function queries the error channel on the disk drive. If any Menu operation results in a blinking light on the disk drive, it

is time to choose this option. It will give you an error number with an English translation, along with the track and sector location of the error if applicable. Error 0 equals no error.



















- **(R) Rename** This function allows you to change the name of any file. It prompts you for an old and new filename, then changes the old filename to the new filename.

- **(W) Write menu** This function automatically puts a copy of the Menu program itself out to disk. After you have formatted a disk, go right to this option and save the Menu program to the disk. Then you can use the new disk directly to access the Menu program from that point forward. You can also put a copy of the Menu on all existing data disks with at least nine remaining sectors. The program takes up only 2.3K and can fit on even the most crowded of disks. Try to find space for it.

- **(Q) Quit Menu** This function eliminates the Menu program from memory and brings you back out to Basic.

The table that follows will help you locate all the special graphics characters you will need to find in order to type in the program:

Disk menu special function keys.

Char/function	Keypress	Line(s)
 clear	 	210
 rvs	 	262,276
 rvs off	 	262
 red	 	225,262
 green	 	225,276
 blue	 	210,225,230,262,282

And here is the program itself:

```

1 REM PROGRAM 1
2 REM DISK USER'S MENU
3 REM
4 REM-----
10 POKE 53280,1:POKE 53281,1
20 GOSUB 200
30 PRINT"-----"
   "
40 PRINT"(<) RUN, (F)ORMAT, (C)OPY, (E)R
ASE"
50 PRINT"(D)IRECTORY, (*) CHECK ERROR ST
ATUS"
60 PRINT"(R)ENAME, (W)RITE MENU, (Q)UIT
MENU"

```

```

70 PRINT"-----
-----"
80 INPUT X$
90 IF X$="D" THEN GOSUB 200
100 IF X$="F" THEN GOSUB 300
110 IF X$="C" THEN GOSUB 400
120 IF X$="E" THEN GOSUB 500
130 IF X$="W" THEN GOSUB 600
140 IF X$="Q" THEN GOSUB 350
150 IF X$="*" THEN GOSUB 650
160 IF X$="←" THEN GOSUB 550
170 IF X$="R" THEN GOSUB 450
180 GOTO 30
190 PRINT:PRINT READ DIRECTLY
200 PRINT:PRINT
210 PRINT:PRINT:PRINT"-----
-----"
220 PRINT"MENU 1541 -- VERSION 2.2 -- C-
64 -- JJA"
223 PRINT"-----
-----"
225 PRINT"LENGTH  NAME
TYPE"
230 PRINT"-----
-----"
251 OPEN 1,8,0,"$"
252 GET #1,A$,B$
254 GET #1,A$,B$
256 GET #1,A$,B$
258 C=0: IF A$(">)" THEN C=ASC(A$)
260 IF B$(">)" THEN C=C+ASC(B$)*256
262 PRINT"  MID$(STR$(C),2);TAB(7);"
";
264 GET #1,B$: IF ST(">") THEN 282
266 IF B$(">")CHR$(34) THEN 264
268 GET #1,B$: IF B$(">")CHR$(34) THEN PRINT
B$;:GOTO268
270 GET #1,B$: IF B$=CHR$(32) THEN 270
272 PRINTTAB(29);:C$=""
274 C$=C$+B$:GET #1,B$: IF B$(">)" THEN 274

276 PRINT"  LEFT$(C$,3)
280 IF ST=0 THEN 254
282 PRINT"BLOCKS FREE"
284 CLOSE 1:RETURN
300 REM FORMAT DISK
305 PRINT"INSERT DISK TO BE FORMATTED.":
PRINT
310 PRINT"INPUT DISK NAME" :INPUT DISK$
320 PRINT "INPUT DISK NUMBER":INPUT EXT$

325 MACRO$="N:"+DISK$+"", "+EXT$
330 OPEN 15,8,15,MACRO$

```

```
340 CLOSE 15:MACRO$="":RETURN
350 REM EXIT PROGRAM
380 PRINT"EXIT TO BASIC.":NEW
400 REM COPY FILE
410 PRINT"INPUT SOURCE FILE NAME" :INPUT
    DISK$
420 PRINT "INPUT NEW FILE NAME":INPUT NW
    S$
425 MACRO$="C:"+NWS$+"="+DISK$
430 OPEN 15,8,15,MACRO$
440 CLOSE 15:MACRO$="":RETURN
450 REM RENAME FILE
460 PRINT"INPUT OLD FILE NAME" :INPUT DI
    SK$
470 PRINT "INPUT NEW FILE NAME":INPUT NW
    S$
475 MACRO$="R:"+NWS$+"="+DISK$
480 OPEN 15,8,15,MACRO$
490 CLOSE 15:MACRO$="":RETURN
500 REM DELETE FILE
510 PRINT"INPUT FILE NAME TO DELETE":INP
    UT DISK$
520 PRINT"HIT <RETURN> TO DELETE":INPUT
    X$
530 MACRO$="S:"+DISK$
535 OPEN 15,8,15,MACRO$
540 CLOSE 15:MACRO$="":RETURN
550 PRINT"TYPE IN FILENAME TO RUN, HIT <
    RETURN> "
570 INPUT N$:LOAD N$,8:RUN
600 REM SAVE MENU FILE
610 PRINT"INSERT DISK TO BE WRITTEN TO."
    :PRINT
620 PRINT"HIT <RETURN> TO WRITE MENU FIL
    E":INPUT X$
625 OPEN 1,8,15
630 SAVE "MENU",8
635 CLOSE 1
640 RETURN
650 REM CHECK ERROR STATUS
650 OPEN 1,8,15
660 INPUT#1,A,B$,C,D
680 PRINT"ERROR STATUS":PRINT:PRINT"ERRO
    R # ";A
685 PRINT B$:PRINT"TRACK ";C,"SECTOR ";D
690 CLOSE 1:RETURN
```

Don't worry if you don't understand all of the commands possible from Menu just yet. You can use as little of it or as much of it as you want. It certainly makes formatting and file maintenance much less of a chore.

■■■■■■■■■■

## DRIVING THE WEDGE

Another program that can help make the disk drive easier to use is the Wedge. This program now ships on the demo disk with all new disk drives.

Loading the Wedge as it appears on the demo disk accompanying new 1541s is as simple as typing

```
LOAD "C-64 WEDGE",8 RETURN
```

Then type

```
RUN RETURN
```

If you have the Wedge supplied on the Commodore Disk Bonus Pack, the loading procedure is slightly different. Type

```
LOAD "DOS WEDGE",8,1 RETURN
```

When the Commodore comes back with a READY, type

```
SYS 52224 RETURN
```

and the Wedge header should come up on the screen. Don't forget the leading space in the filename, or you will get a FILE NOT FOUND error.

Once the Wedge is loaded, disk commands are much easier to invoke. Here is a table of available commands:



## Dos Wedge command format.

Command	Function
↑ <FILENAME>	load, then run <FILENAME>
← <FILENAME>	save <FILENAME>
/ <FILENAME>	load <FILENAME>
% <FILENAME>	load <FILENAME> at address
@	query disk error channel
@\$	list directory
@S:<FILENAME>	scratch <FILENAME>
@I	initialize drive
@UI	reset DOS
@N:<DISKNAME>,<ID>	format disk <DISKNAME>,<ID>
@C:<NEWFILE>=<OLDFILE>	rename <OLDFILE> as <NEWFILE>
@Q	quit DOS wedge

No longer must you memorize cryptic command codes to load, save, scratch, or rename files, or to format disks. Using the Wedge you may call up a disk directory without clearing the current program from memory. You can query the disk error channel with a single keystroke and you can even load and run programs in a single step.





# THE KEY TO THE COMMODORE KEYBOARD

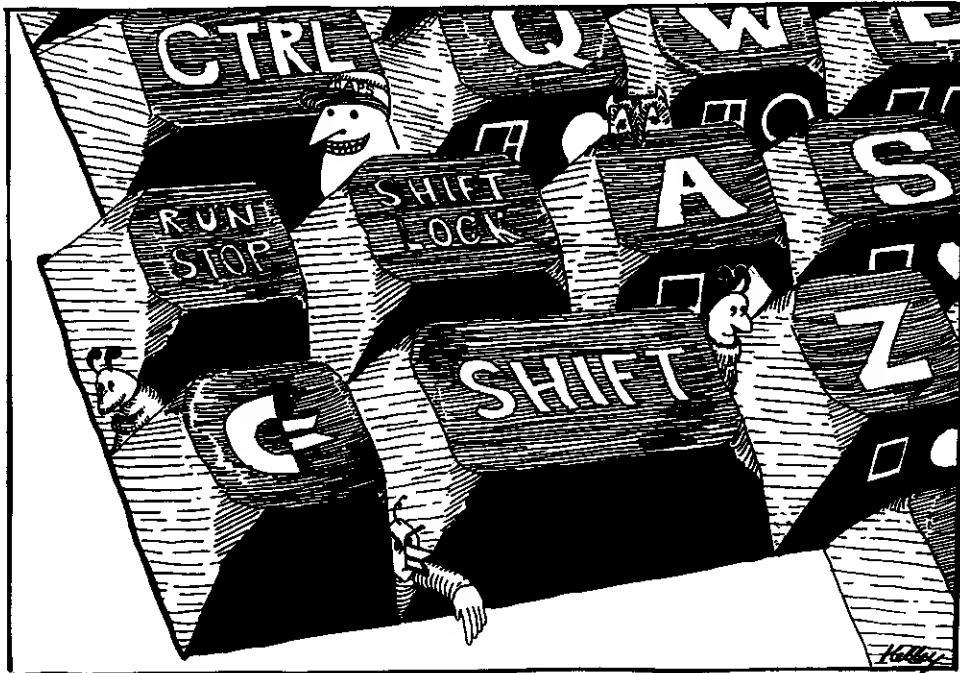
Of course you can make it do plenty of things without learning all that much about its capabilities. You might choose to learn only the commands that load prepackaged software into the machine. But if you want to do more than this, you will have to learn more about the Commodore keyboard layout.

It seems that the questions asked most often about the C-64 concern the keyboard, so we'll hold a short tutorial on the subject. Even if all that follows seems very simple to you, try reading it through. It might help clarify things in the long run.

## THE CAPS MODE

The other mode you can choose is "lower case mode," which is turned on and off by pressing the SHIFT and Commodore logo () keys simultaneously. The screen characters will then resemble those on an ordinary typewriter, with capital letters appearing when SHIFT is pressed.

To get the graphics characters on the lefthand side of each key front, press the Commodore logo key () along with the key that depicts the shape you want. That's all there is to it.



But not every graphics character is depicted on the keyboard itself. There are some mysterious but especially powerful ones you should and will soon get to know.

The reason problems occur if you program in the lower case mode will become obvious if you look at mixed caps and lower case text from the caps mode. All the capital letters will have reverted to graphics characters. Lots of time can be wasted trying to load a filename with caps that have been entered from the lower case mode. Filenames with graphics characters in them are rather inconvenient to type, and since a filename entered in caps from the lower case mode will appear only as graphics characters from the caps mode, it can become terribly frustrating.

This problem may also drive you crazy when you're looking for graphics characters when typing in a program. Stay in the caps mode, unless there is a very good reason to go to lower case. A good rule of thumb is to use the lower case mode when running, as opposed to editing a program. If you are editing lower case text, shift back regularly.

\*\*\*\*\*

## THE QUOTE MODE

You may also have noticed that something funny happens after you type a quotation mark. When the Commodore editor "sees" a quote, it puts the computer into what in other machines is called an "escape" mode. That means that rather than executing a keyboard command, it puts the message to execute that command into a PRINT statement within the very program you are typing. When it sees a closing quote, it reverts back to normal.

Outside a quote, CTRL-BLK will turn the cursor, and all subsequent characters, black. Inside a quote, a special character will be inserted, which tells the machine to turn the cursor black upon execution of that line.

These are where all the mysterious special characters come into play when typing program listings. It takes some practice to get to know these

































*outside key" can  
put a special character in the program*

characters when you see them, let alone what they mean and how to attain them.

Many characters act just like Basic program instructions to complete some screen operation. They change colors, clear the screen, move the cursor, and check the function keys.

Don't feel like you have to memorize all the special function characters, but do become familiar with them. All of the important ones appear in the following table. Try to get to the point where you can at least recognize them.

### C-64 special graphics characters.

	clear	SHIFT	CLR/HOME
	home	CLR/HOME	
	left	SHIFT	←CURSR→
	right	←CURSR→	
	up	SHIFT	↑CURSR↑
	down	↑CURSR↑	
	rvs	CTRL	9
	rvs off	CTRL	0
	black	CTRL	1
	white	CTRL	2
	red	CTRL	3
	medium blue	CTRL	4
	purple	CTRL	5
	green	CTRL	6
	blue	CTRL	7
	yellow	CTRL	8
	orange	↵	1
	brown	↵	2
	pink	↵	3
	dark grey	↵	4
	medium grey	↵	5
	light green	↵	6
	light blue	↵	7
	light grey	↵	8
	f1	f1	
	f2	SHIFT	f1
	f3	f3	
	f4	SHIFT	f3
	f5	f5	
	f6	SHIFT	f5
	f7	f7	
	f8	SHIFT	f7

#### ■■■■■■■■■■

## **Other Things to Know About the Keyboard**

If you want to stop a program during execution, press the RUN/STOP key. If you want to stop a program, clear the screen, and return to default colors, hold down RUN/STOP and press RESTORE.

The SHIFT LOCK key will lock you into upper case. It works in both the caps mode and lower case mode, although chances are that the only time you would want to use it is from lower case mode.

The four function keys have eight labeled functions. The functions f1, f3, f5, and f7 are obtained just by pressing the labeled keys themselves. The function keys f2, f4, f6, and f8 are obtained by holding the SHIFT key down while pressing on a function key.



# BASIC GRAPHICS & SOUND

Okay. You're hooked up now, and you're pumping a high quality signal to your TV or dedicated monitor. You're somewhat familiar with how the keyboard works and how to save and load programs. Now you are ready to begin to make graphics happen on the screen.

We're assuming here that you know nothing at all about computers. If you already know something about Basic, what follows will seem very simple to you, at least at the beginning. It might be a good idea however to go through the paces anyway. We are going to build everything we learn on knowledge we have gained before. So stick with it, even if it seems simple.

For many of you, especially the total beginners for whom this book was designed, the following material will not seem easy. Don't get discouraged. Plod onward, even when commands and techniques don't seem utterly obvious. Things will come together with practice. Sometimes insights work backwards, and a more advanced application will shed light on some simpler concepts that you didn't quite catch the first time.

It is a good idea to save each program as you go—once it is typed in, debugged, and running as advertised. That way you can go back and play with it again when the mood strikes, without having to re-enter anything. If you create your own customized versions of the programs given here, save those too. The idea is to use these seed programs to build bigger and better ones and eventually build up your own personal program library.

Enough of this. Let's get to the fun stuff.

■■■■■■■■■■

## BASIC BASICS

When you turn on the computer, it indicates that everything is in working order by "signing on." It tells you it is running Commodore 64 Basic version two, and that 38,911 bytes are free for Basic programming. You are going to learn how to program those bytes to get the computer to do neat things. If the computer doesn't start off with this message, you'd better go back to the section on setting up.

■■■■■■■■■■

## Deferred and Direct Commands

Don't be put off by the terminology of "deferred" and "direct"—these terms are easily explained. There are two ways of entering commands into Basic. The way we usually think of is the *deferred* method, which involves the use of a legal Basic command with a line number in front of it. This is the

method by which we program in Basic. All Basic programs are made up of lines, and all those lines must be numbered. The order of line numbers in a program tells the computer what order to use when executing commands while running that program. It starts with the lowest line number and works its way up. This is the heart and soul of Basic.

But there is another way to enter commands, and that is in the *direct* mode. If you type a legitimate command without a line number, the computer will try to execute it right then and there—it won't matter whether or not there is a program in memory. You can use a direct command to get the computer to do something you want it to, or even to ask it a question. For example, try typing

```
PRINT FRE(1) RETURN
```

The computer will tell you how much memory is left for Basic programming.

Throughout this book, you will be entering commands in Basic through the deferred and direct modes. Each mode has its time and place, and you will eventually get comfortable with using both of them.

■■■■■■■■

## Printing From Basic

Just about the first thing anybody can learn about Basic is how to print to the screen. Because the C-64 has some special powers, the PRINT statement can be used to very good graphic effect. Type

```
PRINT "HELLO" RETURN
```

and the computer does just that. This is an example of a direct command. Next type in the program below, then type

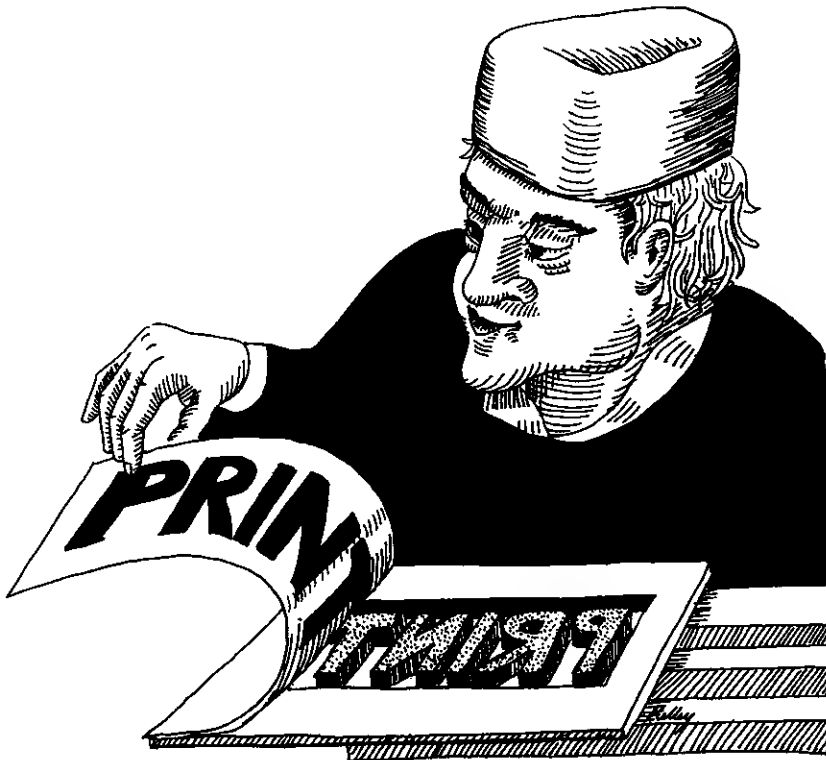
```
10 PRINT "HELLO" RETURN
20 GOTO 10
RUN RETURN
```

This is an example of a deferred command. It is executed only when the program is run. To stop the program, press the RUN/STOP key.

To return to the program type

```
LIST RETURN
```





The command to print is very simply the word PRINT followed by characters enclosed inside quotation marks. For example

```
10 PRINT "HELLO FROM THE DEFERRED MODE"
20 GOTO 10
```

In the program above, line 10 says to print a message on the screen. Line 20 tells the computer to go back and execute line 10 again. Voila! The screen displays the message once again.

After it executes the line again, the computer goes to line 20 for a second time. Line 20 tells the computer to go back to line 10 again. The result is a *loop* which will print HELLO FROM THE DEFERRED MODE again and again on the screen, until we tell the C-64 to stop. To get out of an endless loop, we must press RUN/STOP. This won't hurt anything and it isn't cheating.

And without having to get much more technical than that, we can start to build graphics.

Look at the next program, and try to guess what it will allow us to do.

---

```
1 REM PROGRAM 2
2 REM PRINT TEMPLATE
3 REM KEEP A BLANK COPY ON FILE
4 REM-----
10 PRINT" "
```

```

15 PRINT"
20 PRINT"
25 PRINT"
30 PRINT"
35 PRINT"
40 PRINT"
45 PRINT"
50 PRINT"
55 PRINT"
60 PRINT"
65 PRINT"
70 PRINT"
75 PRINT"
80 PRINT"
85 PRINT"
90 PRINT"
95 PRINT"

```

This program is just waiting for you to enter print characters between the quote marks, so they can be printed out to the screen. If you save this program as it stands, it can act as a *template* or frame for future print graphics. You can use it again and again as a starting point for pictures, which can be saved and loaded from cassette or disk.

Here's a hint on how to type that listing with the least effort from you, and the most from the Commodore 64 editor. Type

```
10 PRINT" {COUNT 29 SPACES} " RETURN
```

Then, using the cursor movement SHIFT-UP ARROW, move the cursor back to the line you just typed. Change the line number from 10 to 15, then hit **RETURN**. A new, duplicate line will automatically be entered. The original line 10 will not be erased or changed in any way. You can do this for each line of the program and save a lot of time and effort in the process. It's like 18 lines for the price of 1!

There are lots of times when you are programming where you can use the cursor keys to make things easier. Get used to using them, and it will pay off in the long run. You're bound to get confused for a while on how to move the cursor—but since it never disturbs anything as it moves over it, you can't harm a program or graphics. And as you get used to working the cursor keys, you'll quickly learn the shortcuts to their most efficient use.

There are a couple of things to bear in mind when using the cursor keys to edit a program. First, the editor acts kind of funny after you enter a quotation mark. Instead of moving the cursor, it prints weird graphics characters. Later we will learn to harness this power to do animation. For now, the phenomenon may turn out to be only a colossal aggravation. Just remember that after you've typed a quotation mark, all cursor movement bets are off, until you either type another quotation mark or hit **RETURN**.

Also, when typing new line numbers over old ones, make sure you don't

try to create two lines with the same number. Luckily, the Commodore 64 is smart enough not to delete an existing line number if you try to create a new one with the same number using the editor as described. But you may be surprised to discover that your new line hasn't been accepted when you give it an existing line number with the editor. Be somewhat careful, and you won't get into serious trouble. At the worst, you'll just have to try again with a new line number. To delete a line, you must type the line number on a fresh line, then press **RETURN**. Or, you can type

**NEW RETURN**

which will clear the entire program out of memory. Don't type NEW unless you don't mind losing all the work you've done up to that point.

Here is the way the cursor moves:

Moving the cursor.

DIRECTION	KEYPRESS
right	<b>←CURSR→</b>
left	<b>SHIFT ←CURSR→</b>
down	<b>↑CURSR↓</b>
up	<b>SHIFT ↑CURSR↓</b>

There is one very important thing to remember about making changes to any program with the screen editor. Merely making the changes, then moving the cursor elsewhere, is not enough. You must hit **RETURN** while the cursor is on any line to send that line to the computer. So make the change, then press **RETURN**. It doesn't matter where the cursor is on the line you wish to enter, just that it is *somewhere* on that line. LIST the program again to see if it has registered the change.

Here's another hint. You can abbreviate the word PRINT by just typing a question mark. After you hit **RETURN**, and then list the program again, the question mark will have "magically" been turned into the word PRINT by the Commodore editor. Once you get used to using the question mark to mean PRINT, you'll never go back to typing the word again.

Now let's get back to our print template. If you run it as it stands, not much happens. The next program uses the template to insert something in between those quotes, and have it printed out to the screen.

---

```

1 REM PROGRAM 3
2 REM THE LETTER FELLA
3 REM SAY HI TO HENRY
4 REM-----
10 PRINT"                                "
15 PRINT"                                "
20 PRINT"                                "
25 PRINT"                                "
30 PRINT"                                "
35 PRINT"                                "
40 PRINT"                                "
45 PRINT"                                "
50 PRINT"                                "
55 PRINT"                                "
60 PRINT"                                "
65 PRINT"                                "
70 PRINT"                                "
75 PRINT"                                "
80 PRINT"                                "
85 PRINT"                                "
90 PRINT"                                "
95 PRINT"                                "

```

---

Typing in the print characters, you may have some trouble with the right quote sign or when you make a mistake. The cursor enters the "quote mode" when it passes a quotation mark, and will behave differently until it encounters another. Remember, you can back out of a line by hitting **RETURN** and then go back and rework it with the cursor keys. Use the cursor keys rather than the INST/DEL key to change the print characters.

If you find yourself messing up the right quotation mark, don't worry about it. It doesn't matter where it is, as long as whatever you want to print is to the left of the second set of marks. In fact, you can leave the second set of quotation marks off completely, and the program will still run.

When the picture is done and you are happy with it, move the cursor back up to line 10 and start hitting **RETURN**. Hit **RETURN** for every line of your program that contains a part of the picture. Otherwise, the program won't know exactly what to remember. Later we'll save pictures in a different way, and you won't have to do this chore.

When you RUN this program, the computer will print out a picture of the "letter fella." He's made entirely of M's, O's, l's, left and right brackets, "greater-than" and "less-than" signs, equal signs, and dashes. While the appearance of our little friend may be a bit primitive, the idea underlying how he was made will eventually allow us to begin creating much more sophisticated images. The PRINT statement can do a surprising amount of graphics work on the Commodore 64.

Take a look at the front of the keys on the C-64, as opposed to the tops of them. See all those weird shapes? Those are special graphics characters, specially designed to draw with. You can type them directly from the keyboard whenever you want. The shapes on the front lefthand side of each key will appear on the screen when you hold down the Commodore logo key and

press another key. In caps mode, the shapes on the front righthand side of each key will appear on the screen when you hold down the SHIFT key and press another key.

A quick word about caps and lower case mode. Hitting the Commodore logo key and SHIFT together toggles you in and out of these states. Hit them together a few times, and see what happens. In the caps mode, all letters are upper case, and alphabetical keys result in graphics characters when SHIFTed. That is the best mode to be in when programming, and certainly the mode to be in when using this book. If you use the lower case mode, then shift to caps mode, all caps will turn to graphics symbols. This can be very messy. Also, Basic is easier to read and write without worrying about lower case. So stick to caps at least for the time being.

Let's rework our little friend using some of the special graphics characters possible on the Commodore 64.

Instead of M's for hair, we'll use the shape of the Commodore logo key and the plus sign (+). For eyebrows we'll use the Commodore logo key and the British pound sign (£). For eyes we'll use SHIFT-W (◉). Instead of brackets for the ears, we'll do them in two parts each: the left ear as SHIFT-U (⌞) and SHIFT-J (⌵), and the right ear as SHIFT-I (⌠) and SHIFT-K (⌡). The sides of the face are made of SHIFT-Ms (⌢) and SHIFT-Ns (⌣). See if you can find the other three shapes on your own. Once you become somewhat familiar with the graphics characters, you'll find that you will easily remember your favorites.

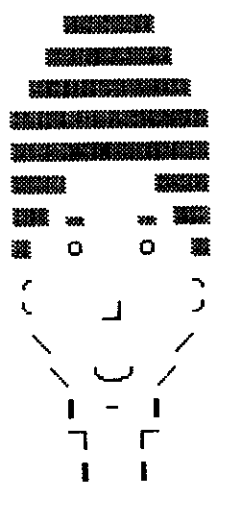
Don't forget to go back and hit **RETURN** over each program line you rework. Otherwise your efforts won't be entered into the memory of the C-64.

---

```

1 REM PROGRAM 4
2 REM BEGINNING GRAPHICS CHARACTERS
3 REM HARRY GOES TO FINISHING SCHOOL
4 REM-----
5
10 PRINT "
15 PRINT "
20 PRINT "
25 PRINT "
30 PRINT "
35 PRINT "
40 PRINT "
45 PRINT "
50 PRINT "
55 PRINT "
60 PRINT "
65 PRINT "
70 PRINT "
75 PRINT "
80 PRINT "
85 PRINT "
90 PRINT "
95 PRINT "

```




---

As you can see, the entry of special graphics characters can improve the look of your work quite a bit indeed (and wait until you see what's next...).

# COLORING

## THINGS IN

Getting the C-64 to work in living color is very easy. Press **RETURN** a few times, then try this: hold the CTRL key down while pressing a number from 1 to 8. The color of the flashing cursor will change. If you begin to type characters or graphics characters with the cursor in a certain color, the characters will print out in that color.

When programming on the 64, try pressing CTRL-2 before doing much else. This turns the cursor, and all subsequent typed characters, a bright white. You may find that the contrast of white characters on a blue background is easier to read and to work with than the tight blue that is automatically provided.

The fronts of the keytops numbered 1 to 8 label the colors that can be obtained by pressing CTRL along with them. By holding down the Commodore logo key and pressing 1 through 8, you can get the second group of eight colors, for a total of sixteen colors. The numbers and keycodes for the sixteen colors are as follows:

### Keypress and color codes.

Color	Keypress	Color	Keypress
0 black	<b>CTRL</b> <b>1</b>	8 orange	<b>⌘</b> <b>1</b>
1 white	<b>CTRL</b> <b>2</b>	9 brown	<b>⌘</b> <b>2</b>
2 red	<b>CTRL</b> <b>3</b>	10 pink	<b>⌘</b> <b>3</b>
3 medium blue	<b>CTRL</b> <b>4</b>	11 dark grey	<b>⌘</b> <b>4</b>
4 purple	<b>CTRL</b> <b>5</b>	12 medium grey	<b>⌘</b> <b>5</b>
5 green	<b>CTRL</b> <b>6</b>	13 light green	<b>⌘</b> <b>6</b>
6 blue	<b>CTRL</b> <b>7</b>	14 light blue	<b>⌘</b> <b>7</b>
7 yellow	<b>CTRL</b> <b>8</b>	15 light grey	<b>⌘</b> <b>8</b>

Play around with color graphics in the direct mode until you've had your fill. Next we shall learn how to enclose color graphics within PRINT statements.

Type NEW **RETURN** to clear any current program from memory. Next, type 10 PRINT" and then try holding down CTRL or the Commodore logo key, while pressing down on a number key from one to eight.

Surprised? The cursor color does not change. Instead, a special graphics character appears on the screen. This is one of the special aspects of the quote mode, which we hinted at earlier. When the C-64 editor encounters a quotation mark, it changes direct commands into special graphics characters that represent those special commands. Then, when the program is executed, those commands are interpreted and acted upon. This is how you can get color pictures to appear on the screen.

---

```

1 REM PROGRAM 5
2 REM COLOR GRAPHICS CHARACTERS
3 REM AND THEIR COLORS
4 REM-----
10 PRINT"■ BLACK"
20 PRINT"□ WHITE"
30 PRINT"■ RED"
40 PRINT"▤ CYAN (LIGHT BLUE)"
50 PRINT"■ PURPLE"
60 PRINT"■ GREEN"
70 PRINT"■ BLUE"
80 PRINT"■ YELLOW"
90 PRINT"■ ORANGE"
100 PRINT"■ BROWN"
110 PRINT"■ PINK"
120 PRINT"■ DARK GREY"
130 PRINT"■ MEDIUM GREY"
140 PRINT"■ LIGHT GREEN"
150 PRINT"■ MEDIUM BLUE"
160 PRINT"■ LIGHT GREY"
READY.

```

---

When typing this program, you can easily find the special characters. They go in order—hold CTRL, then press 1, 2, 3, 4, all the way to 8 before typing the color names. Then hold down the Commodore logo key, and press 1 through 8 again before typing the second eight names. It might be a good idea to get to know the special color characters. Then when you see them in a program, you will recognize what they do.

This program lets us look at all sixteen colors, and see what special graphics characters represent them. Each of the special characters in front of the words that describe the colors themselves tell the computer to print in that color. When the computer encounters those color characters during the RUN of a program that includes them, it automatically switches to the colors when printing whatever follows them.

But wait a minute. When you run this program, there is a space between green and yellow right where dark blue should be. Can you guess why this happens?

The words DARK BLUE are in fact printing out in the color dark blue, but because the background color itself is dark blue, you can't see the printout.

Another thing that may happen on your system when running this program is that some of the words may be hard to read, especially BROWN. Some color combinations on the C-64 work better than others, and brown on a dark blue background is not one of the big winners. The time has come for us to learn how to change background and border colors on the computer.

■■■■■■■■■■

## Changing Colors With POKE

Some of the powers of the Commodore 64 truly seem like magic. Like the sorcerer's apprentice, you've got to be careful harnessing these powers, or things may happen that are beyond your control.

POKE commands are a good example of this. Using the word POKE, you can tell the computer to do some very special things. We will be making

much greater use of POKE commands as we progress. But like many good tools, they can be dangerous if not used correctly.

Try typing POKE 53272,6 RETURN into your computer. This tells the C-64 to put the value 6 into memory location 53272. Watch what happens to the screen display when you do so. What you've done is robbed the C-64 of its normal sense of "identity." The pointer that tells it where to find its usual character set has been changed, so typing anything else will be pretty difficult.

You can still get out of it though. You can carefully type POKE 53272,21 RETURN. Or you can hit RUN/STOP and RESTORE together. Whew. Glad to be out of that predicament.

Try typing POKE 53272,1. This blows the mind of our C-64 in yet another manner. You'll have to restore things again to continue with our discussion.

So we can see that POKes are very powerful commands from Basic.

Unfortunately, the version of Basic that comes packed inside the Commodore 64 does not have special commands to make it do all that we want it to. Up ahead we'll explore the language Simon's Basic, which is much better able to let us get at the power of the C-64. To program graphics and sound from plain old Basic, you're going to have to learn more about POKes, in order to do things plain old Basic can't do.

Location 53281 is just about the first one you should learn to use effectively. A POKE to that location allows us to change the background color on the C-64.

Get yourself back into Basic, and RUN the color program above. There's that space between GREEN and YELLOW. Now type POKE 53281,0 RETURN. Look what happens.





The screen background turns black. Now you can read DARK BLUE, and see that it had in fact printed out when the program ran. Problems with smearing, as with the word BROWN, are eliminated. You can no longer see the word BLACK, though, because it is now printed out on a black background.

Turning the background black when working with color graphics will make the colors look as bright as possible, with no opportunity to smear. To do this, you would have to place the command POKE 53281,0 into the first line of a graphics program. You can easily make the background color into any of the sixteen available.

You can also change the border color just as simply. Its color is stored in location 53280. To change the border color to black, type POKE 53280,0.

### POKE color codes.

0 black	8 orange
1 white	9 brown
2 red	10 pink
3 medium blue	11 dark grey
4 purple	12 medium grey
5 green	13 light green
6 blue	14 light blue
7 yellow	15 light grey

Use the table above to change background and border colors to any you choose. You may or may not want to memorize the actual color values. Note that they do not match the first eight number keys and their colors. When POKEing colors, black is not 1 but 0. White is 1, red is 2, and so on. This may be a bit confusing, but work with it. Forget about the numbers on the color keys themselves.

Two numbers you will want to memorize, however, are 53280 and 53281. Once you have them in your head, using them can become second nature.

■■■■■■■■■■

## Looping in Color

Remember our very simple loop, where line 20 told the computer to go back and execute line 10 again and again? Let's write a program that loops through background and border colors. It's easy.

---

```

1 REM PROGRAM 6
2 REM FLASHING BACKGROUND COLORS
3 REM USING POKES AND LOOPS
4 REM-----
10 X=X+1
```

```

20 IF X>15 THEN X=0
30 POKE 53280,X
40 POKE 53281,X+1
50 FOR I=1 TO 200:NEXT I
60 GOTO 10

```

---

It's only six lines long, but this program includes a couple of concepts that will allow us to go on to do some amazing things. Let's see how it works.

Line 10 introduces the concept of the *variable*, and the concept of the *counter*. The power of variables comes from the fact that they do not need to remain the same fixed number all the time, but can change their values, even in the course of a single program. Sometimes their values change in a single program line, as in line 10 of this program. This line says to take the value of some variable named X, and add 1 to it. Since the computer has not encountered the variable X before, and we have not set it to any special number beforehand, it has assigned no value to X—so X starts off as 0. Then, at line 10, we tell the C-64 to add 1 to it. The machine says to itself that  $0+1=1$ , so  $X=1$ . Therefore, line 10 is a counter for variable X.

■■■■■■■■■■

## IF/THEN

Line 20 is a very powerful command, known as IF/THEN. As you progress with this book, you'll learn about the many things this command can do. The format for this command is as follows:

IF expression THEN command

where:

expression = defined expression

command = legal Basic command

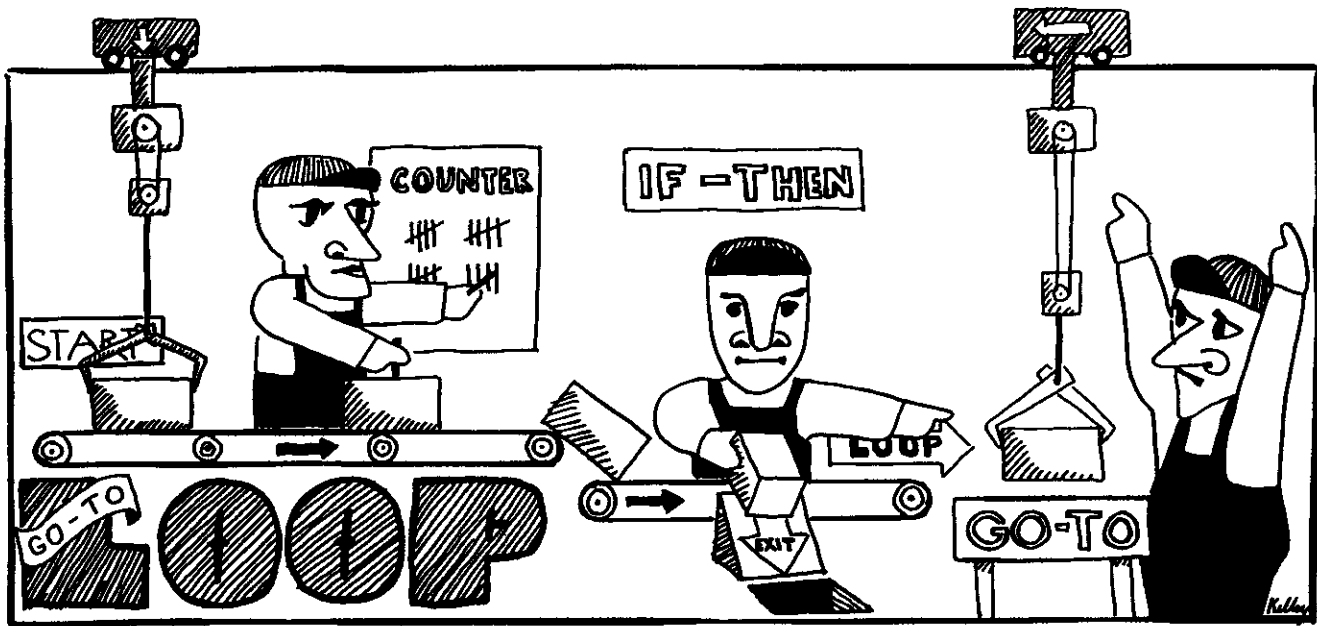
Don't worry if IF/THEN seems a little confusing now. Here all it says is if X is greater than 15, don't let it become 16, but set it back to 0. Using this command, we can shape the value of X to meet our requirements. It is as simple as that. In this program,  $15+1$  will equal 0—because in line 20 we've told the computer that this is the way it shall be.

Aren't computers fun?

When the computer begins the RUN of this program, at line 10 it sets the value of the variable X to 1. At line 20, it asks itself if X is greater than 15. This time through, X is not greater than 15. When the "if" condition of an IF/THEN statement is not met in a program RUN, the computer ignores the "then" completely. So on we go to the next line.

Line 30 should seem somewhat familiar. We will POKE the value of location 53280, which sets the border color. There is a change here, however. Instead of POKEing the location with some fixed number, we shall POKE it with the value of the variable X. The computer is just as happy to set the value of location 53280 to X as it is to any constant. This time through,  $X=1$  and the color of the border will be switched to white.

Line 40 does the same thing for the background color, with one change: it adds 1 to the value of X before executing the POKE. We could very easily have set location 53281 to the same value as 53280. But that would change



## FOR/NEXT

the entire screen to a solid color, and we would lose some effect. What is neat here is to have the background and border colors contrast during program execution. So we POKE the background color to  $X+1$ . When the program runs through the loop the first time, line 40 sets location 53281 to 2.

If you haven't spent any time with Basic, line 50 may seem like a real puzzler. Believe me, by the time you're done with this book, you'll wonder how you ever got along without FOR/NEXT loops.

This whole program is a loop, and line 50 is a loop within the loop. When a program has a line like line 60 here, which says to go back to an earlier line, a loop is constructed.

A FOR/NEXT loop is another way of setting up a loop, and a handy one at that. The format for the use of FOR/NEXT is

```
FOR variable = n to range
NEXT variable
```

where:

variable = variable chosen to count the loops

n = starting value of variable

range = number at which counting stops

The neat thing about FOR/NEXT loops is that you can get them to count for you, and they will stop automatically at the number you specify. You tell the computer how many times you want it to loop. It will count each time through, and then let you out of the loop. We'll be talking more about this up ahead.

But why is line 50 necessary here, and what does it do? Well to answer that question, you might first try running the program without line 50. Go ahead.



Line 20 never does us any good until the 15th time through the loop. Then variable X reaches line 10 with a value of 15, and line 10 says add 1 to it. The computer reaches line 20 with X equaling 16. Now line 20 does its stuff. The value of X *is* greater than 15, so the THEN side of the IF/THEN statement is activated. X remains 16 only for a very short time. Line 20 then turns the value of X back to 0.

This way we can reset the values to be POKEd into the border and background locations. What happens if we remove line 20? Actually, the program will still run—for a while. The C-64 will accept numbers up to 255 in locations 53280 and 53281 without ill effect. Once we attempt to POKE a value of 256 in either location, however, the program crashes with an ILLEGAL QUANTITY error. Line 20 allows the program to run as long as we want, and until we press the RUN/STOP key.

You may be finding this material to be quite easy to understand. If you are, good, and rest assured, we're well on the way to the good stuff. If you're having a problem grasping the material presented so far, study this program and pretend to be the computer. Go through the loop each time in your mind. Play with the values of the program. Get a grip on how it works. You'll need it for what's to come.

■■■■■■■■■■

## Color Graphics From Print Statements

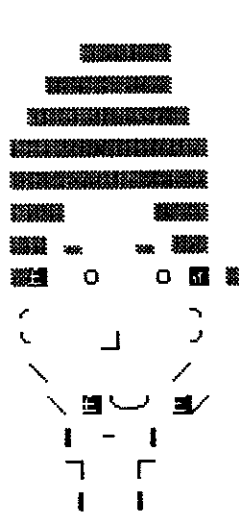
Remember our portrait? Let's get our friend to print out in living color.

---

```

1 REM PROGRAM 7
2 REM HARRY GETS SOME COLOR
3 REM IN HIS FACE
4 REM-----
5 POKE 53281,0
10 PRINT"
15 PRINT"
20 PRINT"
25 PRINT"
30 PRINT"
35 PRINT"
40 PRINT"
45 PRINT"
50 PRINT"
55 PRINT"
60 PRINT"
65 PRINT"
70 PRINT"
75 PRINT"
80 PRINT"
85 PRINT"
90 PRINT"
95 PRINT"

```




---

First we get the background to turn black. That's what line 5 does. But wait—our buddy's face looks a bit messed up when viewed as a list. What the heck is going on here?

In line 10, we turn the cursor yellow. It will remain yellow until we tell it otherwise. That's why the hair comes out blond, all the way through to line 50. But things seem to go haywire at line 50. The eyes are out of place. The right sideburn is doubly out of place. The mouth also seems wrong.

Type in the program the way it appears, and you'll see that it prints out correctly when RUN. Our picture is exactly the way it was before, only now it is in color.

When we insert special graphics characters in a program listing to change cursor color, they will not be printed out as they appear, yet they take up spaces in the listing. Because of this, you must make up for the space they occupy by adding an extra space for each one you use—before typing the characters that will be printed out.

In line 50, we first want to make the eyes red. So we type the character that indicates a change to red in the print statement.

Then, to make up for the space that the graphics character itself takes up, we must add a space to print before the eyes. This is why in the program listing the eyes are pushed one space to the right of where it seems they should be. By the time we get to the right sideburn, we want to go back to yellow, and so we enter a second color change character. Now we must insert another space, which pushes the sideburn *two* spaces away from where it may seem to belong.

When the program runs, the color change characters will not print out, and what is left will fall back into the order we intend it to be.

This displacement problem is one of the major disadvantages of the print template approach to color graphics, but you can learn to work with it. All you need to do is get the hang of adding spaces where necessary. Here are some examples of one-color shapes and the changes the listings undergo when color is added.

---

```

1 REM PROGRAM 8
2 REM MONOCHROME TO COLOR
3 REM BEFORE AND AFTER
4 REM-----
5 REM "BEFORE "
10 PRINT"  {
20 PRINT" |3 |
30 PRINT" | ♥ |
40 PRINT" | ♥ |
50 PRINT" | ♥ |
60 PRINT" |   3 |
70 PRINT" }
95 REM "AFTER "
110 PRINT"■ {
120 PRINT" |3■ |
130 PRINT" | ♥■ |
140 PRINT" | ♥■ |
150 PRINT" | ♥■ |
160 PRINT" |   3■ |
170 PRINT" }

```

---



---

```

1 REM PROGRAM 9
2 REM MONOCHROME TO COLOR
3 REM BEFORE AND AFTER
4 REM-----
5 REM "BEFORE "

```

---

```

10 PRINT "┌───┐"
20 PRINT "│A  │"
30 PRINT "│    │"
40 PRINT "│  ▲  │"
50 PRINT "│    │"
60 PRINT "│   A │"
70 PRINT "└───┘"

95 REM "AFTER"
110 PRINT "┌───┐"
120 PRINT "│■A  │"
130 PRINT "│    │"
140 PRINT "│  ■▲  │"
150 PRINT "│    │"
160 PRINT "│   ■A │"
170 PRINT "└───┘"

```

---

```

1 REM PROGRAM 10
2 REM MONOCHROME TO COLOR
3 REM BEFORE AND AFTER
4 REM-----
5 REM "BEFORE"
10 PRINT " MONOCHROME SQUARES"
20 PRINT " ┌──┐ ┌──┐ ┌──┐"
30 PRINT " │ │ │ │ │ │ │"
40 PRINT " └──┘ └──┘ └──┘"
50 PRINT " ┌──┐ ┌──┐ ┌──┐"
60 PRINT " │ │ │ │ │ │ │"
70 PRINT " └──┘ └──┘ └──┘"
95 REM "AFTER"
110 PRINT "■ ┌──┐ ┌──┐ ┌──┐ ■"
120 PRINT "■ │ │ │ │ │ │ │ ■"
130 PRINT "■ └──┘ └──┘ └──┘ ■"
140 PRINT "■ ┌──┐ ┌──┐ ┌──┐ ■"
150 PRINT "■ │ │ │ │ │ │ │ ■"
160 PRINT "■ └──┘ └──┘ └──┘ ■"
170 PRINT "■ ┌──┐ ┌──┐ ┌──┐ ■"

```

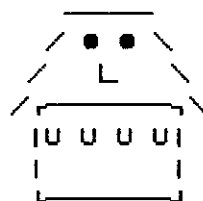
---

```

1 REM PROGRAM 11
2 REM MONOCHROME TO COLOR
3 REM BEFORE AND AFTER
4 REM-----
5 REM "BEFORE"
10 PRINT "      ┌───┐"
20 PRINT "      │   │"
30 PRINT "      │   │"
40 PRINT "      │   │"
50 PRINT "      │U U U U│"
60 PRINT "      │       │"
70 PRINT "      └───┘"
95 REM "AFTER"

```

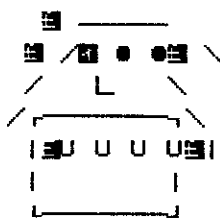
---



```

110 PRINT"
120 PRINT"
130 PRINT"
140 PRINT"
150 PRINT"
160 PRINT"
170 PRINT"

```



As you may have noticed by now, another disadvantage of this approach is the limitation of the size of your shapes. It gets confusing to try and design a shape wider than 30 characters because you won't be able to clearly envision your shape while composing it. Though a program line can be up to 80 characters long, this approach keeps them much shorter, so that what you see will be at least something like what you get. Up ahead we will learn a trick that allows us to design, save, and retrieve pictures the size of the whole screen quickly and easily, though saving and retrieving will take a bit longer than the fraction of a second this method takes.

The advantages of printing color graphics in this way are the simplicity and speed with which the graphics can be saved and retrieved. After we learn a few more tricks, we will even learn to animate shapes using this approach.

■■■■■■■■■■

## The REVERSE Mode

Here's another powerful trick. First, make the background color black—type POKE 53281,0. Next, press CTRL and while holding it down, type 9. On the front of the 9 key you'll notice the mysterious phrase RVS ON. Type whatever you want, and look at the screen.

Instead of characters appearing the way they usually do, the letters will be formed in the background color, on a background square of the cursor color. This is the *inverse* or REVERSE mode. It is as if printing takes place in a stencil of the normal characters.

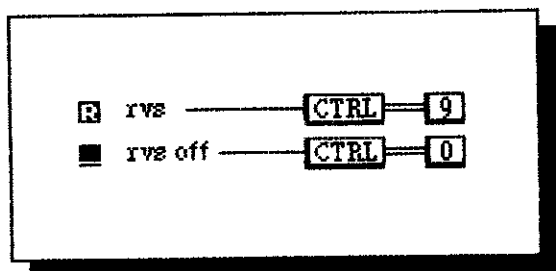
To get out of this print mode, press CTRL-0. The front of the 0 keytop is labeled RVS OFF, and when pressed with CTRL, it does just that—it turns off the REVERSE mode. So does hitting **RETURN**.

Now it is very simple to change color in the reverse mode, too. Press CTRL-9, then hit some keys, then change the cursor color in the way you've learned, then hit some more keys, and so on. Take a look at what happens when you hit the spacebar in the reverse mode. You get a solid square of whatever color the cursor is set to.

Using the reverse mode and the spacebar is just about the simplest way there is to fill the screen with color. Like the cursor color codes, special graphics characters can be inserted into PRINT statements to turn the reverse mode on and off.



## Inverse character codes.



Just as with cursor color changes, turning the reverse mode on inside quotation marks produces a special character, which will turn the reverse mode on in deferred mode when that program line is reached during a RUN. Like with the cursor characters, the presence of the RVS character will displace the print characters by one space.

There is one big difference however in the way RVS acts as opposed to the way the cursor colors act. When RVS is on, it acts only on the one program line in which it appears. When we colored the face picture, all we had to do was turn the cursor yellow once, and it remained yellow through four program lines. The reverse mode must be turned on in every program line it is to act upon.

---

```

1 REM PROGRAM 12
2 REM RAINBOW
3 REM USING RVS ON AND COLOR CODES
4 REM-----
10 POKE 53281,0
20 PRINT"■"
30 PRINT"■"
40 PRINT"■"
50 PRINT"■"
60 PRINT"■"
70 PRINT"■"
80 PRINT"■"
90 PRINT"■"
100 PRINT"■"
110 PRINT"■"
120 PRINT"■"
130 PRINT"■"
140 PRINT"■"
150 PRINT"■"
160 PRINT"■"
  
```

---

If we had not inserted a RVS ON character in each line here, only the first one would print as a color bar. By inserting one in each program line, we get a rainbow-colored test pattern.

## REFERENCES

\_\_\_\_\_

\_\_\_\_\_

By sticking a command or commands in between the FOR and the NEXT statements, we can make the computer execute these instructions however many times we like. Here, we have said to count from 1 to 23, and then inserted a PRINT statement between the FOR and the NEXT statements. What will happen when we run this program?

But before you see the NEXT statement, you are told by line 20 to PRINT something. REVERSE is turned on, and the cursor color is switched through every possibility. So you print that line to the screen.

The process happens again and again, until the value of X reaches 23. Then the FOR/NEXT loop terminates, and the program ends.

Hold everything a second. Isn't there something wrong with line 30? Instead of saying NEXT X, all it says is NEXT. But the program still works. Why?

\*\*\*\*\*

**EASY ANIMATED  
GRAPHICS**

statement inside another. In that situation, you must always tell the computer which loop you are talking about when you type NEXT. For now, though, you don't need to.

Time to pull a really neat trick out of the hat. Try this: put a semicolon after the final quote in the print statement (line 20) in the previous program. Witness the result by running the program.

Originally, each time through the loop, the program printed line 20 on a new screen line, making an ordered test pattern. By putting a semicolon in after the final quote in line 20, you command the computer to start the next print line right where the last one left off. Instead of each print line in the loop starting off on the lefthand side, each starts right at the point where the last one ended. Our test pattern becomes a diagonal mosaic pattern.

By taking things one step further, we can obtain very pleasing animated abstract patterns. Try this one on for size.

---

```

1 REM PROGRAM 14
2 REM EASY ANIMATED GRAPHICS
3 REM
4 REM-----
5 POKE 53281,0:POKE 53280,0
10 PRINT"  █  █  █  █  █  █  █  █  █
   █  █  █  █  █  █  █  █  █
20 GOTO 10

```

---

This program is very much like the two we've just looked at—the print line is just the same, with the addition of the semicolon at the very end. However, the screen has now been turned totally black by line 5, and the FOR/NEXT loop has been replaced by an endless GOTO loop.

When we run the program, the color bars print continuously, until we press the RUN/STOP key. As the bars print to the bottom of the screen, the print scrolls upward. The effect is a looping cascade of color. Let's see any other computer do this quite so easily!

You can harness the power of scrolling PRINT statements for a sophisticated effect. By inserting graphics characters as well as RVS bars, you can create textured effects. The programs themselves can be very short—you can, in fact, fit them on a single program line. Here are some examples to get you going.

---

```

1 REM PROGRAM 15
2 REM
3 REM
4 REM-----
10 PRINT"███  █  █  █  █  █  █
   █  █  █  █  █  █  █  █  █
20 GOTO 10

```

---

```

1 REM PROGRAM 19
2 REM ANOTHER SCROLLING COLOR EXAMPLE
3 REM
4 REM-----
10 POKE 53280,0:POKE 53281,0
20 PRINT "XXXXXXXXXXHLXIXX      XX--X-
-X-XXXXXXX0000000000000000000000";
30 GOTO 20

```

Using FOR/NEXT loops, you can stack print lines in a single program to move through different patterns at whatever rate you desire.

```
1 REM PROGRAM 20
2 REM LOOPING THROUGH TWO
3 REM ANIMATED PATTERNS
4 REM-----
10 FOR X=1 TO 100
```

```

20 PRINT "#####\#####|
|#####|#####";NEXT
30 FOR X=1 TO 100
40 PRINT "#####|#####
#####";NEXT
50 FOR X=1 TO 100
60 PRINT "#####\#####-#####
#####";NEXT
70 GOTO 10

```

---

Here each FOR/NEXT loop runs through a different pattern, then moves on to the next. These loops are enclosed within a GOTO loop, which starts everything over again. See how easy it is to nest loops? There's nothing to it.

Want your abstract pattern to go truly nuts? Poke the value of X into the background location each time through the loop. Add the statement POKE 53281,X as lines 15, 35, and 55 in the program above.

```

1 REM PROGRAM 21
2 REM FLASHING THE BACKGROUND
3 REM DURING EASY ANIMATION
4 REM-----
10 FOR X=1 TO 100
15 POKE 53281,X
20 PRINT "#####\#####|
|#####|#####";NEXT
30 FOR X=1 TO 100
35 POKE 53281,X
40 PRINT "#####|#####
#####";NEXT
50 FOR X=1 TO 100
55 POKE 53281,X
60 PRINT "#####\#####-#####
#####";NEXT
70 GOTO 10

```

---

Talk about psychedelic! A calmer multicolor background may be preferable in the long run.

Here is one to experiment with.

```

1 REM PROGRAM 22
2 REM FLASHING THE BACKGROUND
3 REM AT A MORE REASONABLE RATE
4 REM-----
10 X=X+1
20 IF X>20 THEN Y=Y+1

```

```
23 IF X>20 THEN X=0
25 IF Y>15 THEN Y=0
30 PRINT "*****";
40 POKE 53281,Y
50 GOTO 10
```

Here we have taken the concept of the counter one step further. We have set up a counting GOTO loop much like the one we used earlier, using IF/THEN statements to keep the values of X and Y trimmed to our requirements. The major difference this time is that we have set up two counters—and one counts much slower than the other. As we move through the loop, Y counts by 1 only after X has counted by 20. If we were to POKE location 53281 with the value of X instead of Y, the background color would change as fast as it did in our psychedelic program. But Y counts more slowly, allowing the background color to change at a much more restful pace.

Line 10 sets up the X counter, just as it did in our earliest print program. Line 20 says that when X has counted to 20, Y can count once. Only if X is about to become 21 can Y increase by one.

Line 23 says to reset X to zero once it has counted past 20. Then it starts counting all over again. Line 25 says to reset Y when it reaches past 15. Just as in our color changing program, we want to go back to black once we have cycled through all the available colors.

Line 30 contains our abstract pattern. Don't forget the semicolon after the closing quote mark. Line 40 sets the background color to the value of Y. Line 50 sends us back to the beginning, to start all over again. We add one to X, and go through the loop again.

**That's all there is to it!**

Now we'll discuss some special graphics characters you can use for animation. They alter the screen display by moving the cursor, or by clearing the screen. Using them, we can make shapes move across the screen.

Pressing SHIFT-CLR in the direct mode has the effect of clearing the screen. You can include this special graphics character, like all others, between quotes in a print statement to clear the screen during a program run as well. The format is as simple as

PRINT "SHIFT-CLR" RETURN

where we obtain the special graphics character by typing SHIFT-CLR. This command is used often at the top of a program to clear the screen so that printing can take place.

You can also use this function to perform a simple, if flickery, form of animation. Here is a very crude example:

---

```

1 REM PROGRAM 23
2 REM CRUDE SHIFT-CLEAR ANIMATION
3 REM
4 REM-----
10 PRINT"□"
20 PRINT"●";
30 PRINT"□"
40 PRINT" ●"
50 PRINT"□"
60 PRINT"  ●"
70 PRINT"□"
80 PRINT"   ●"
90 PRINT"□"
100 PRINT"    ●"
110 PRINT"□"
120 PRINT"     ●"
130 PRINT"□"
140 PRINT"      ●"
150 PRINT"□"
160 PRINT"       ●"
170 PRINT"□"
180 PRINT"        ●"
190 PRINT"□"
200 PRINT"         ●"
210 GOTO 10

```

---

In this program, we position a shape, print it, clear the screen, then print the shape one space to the right. We repeat the process a few times, and the shape appears to be moving to the right.

Let's take it one step further. We'll design a program that prints a more complex, multi-line shape, then move that shape across the screen in the very same way.

---

```

1 REM PROGRAM 24
2 REM VARIATION ON THE SAME THEME
3 REM
4 REM-----
10 PRINT"□"
20 PRINT"  ┌─┐  "
30 PRINT"  │ │ │  "
40 PRINT"  └─┴─┘  "
50 PRINT"□"
60 PRINT"    ┌─┐  "
70 PRINT"    │ │ │  "
80 PRINT"    └─┴─┘  "
90 PRINT"□"
100 PRINT"      ┌─┐  "
110 PRINT"      │ │ │  "
120 PRINT"      └─┴─┘  "
130 PRINT"        ┌─┐  "
140 PRINT"        │ │ │  "

```

---

```

150 PRINT"      "
160 PRINT"      "
170 PRINT"      "
180 PRINT"      "
190 PRINT"      "
200 PRINT"      "
210 PRINT"      "
220 PRINT"      "
230 PRINT"      "
240 PRINT"      "
250 PRINT"      "
260 PRINT"      "
270 PRINT"      "
280 PRINT"      "
290 PRINT"      "
300 GOTO 10

```

But there is even a better way to effect animation: using special cursor-movement graphics characters. There are codes to move the cursor one place left, right, up, or down, and they can be used in any combination you like. They can be placed inside print statements, and when used in conjunction with spaces, they act to erase old print positions in the course of a sequence of commands or program loop.

## INVERSE BRACKET II

Probably the handiest of the cursor movement characters is INVERSE BRACKET, obtained by pressing SHIFT-CRSRLEFT while in the quotation mode. The format is

```
PRINT "SHIFT-CRSRLEFT"
```

and you can see that the cursor left graphics character is composed of two small rectangles side by side.

Type in the following program and see how straightforwardly it erases old print positions as it creates new ones.

```

1 REM PROGRAM 25
2 REM AN EXAMPLE OF ANIMATION
3 REM USING CURSOR MOVEMENT KEYS
4 REM-----
5 PRINT"      "
10 PRINT"      "
20 FOR X=1 TO 50:NEXT
30 GOTO 10

```

Now look at this program and see if you can guess how it employs SHIFT-CRSRLEFT to take care of erasure:



```

1 REM PROGRAM 26
2 REM ANOTHER EXAMPLE OF ANIMATION
3 REM USING CURSOR MOVEMENT KEYS
4 REM-----
5 PRINT"␣"
10 PRINT"██ 00";
20 FOR X=1 TO 50:NEXT
30 GOTO 10

```

Each time the program prints out a new position for the ball, it first moves left and prints a blank space over the old position in which the ball appeared. Then it moves right two spaces, to print the ball at a new position. The animation is smooth and relatively flicker-free.

You can animate longer shapes by using multiple cursor movement. Here is an example:

```

1 REM PROGRAM 27
2 REM AN EXAMPLE OF SHAPE ANIMATION
3 REM USING CURSOR MOVEMENT KEYS
4 REM-----
5 PRINT"␣"
10 PRINT"  ██████  ██████  L-O
-J██████O";
20 FOR X=1 TO 50:NEXT
30 GOTO 10

```

## Saving Low-Res Screens

Here is a program that allows you to use the disk drive to save and retrieve low-res screens you have designed. A chart is included to help you find all the special graphics characters you need to type in the listing.

Screen save special function keys.

Char/function	Keypress	Line(s)
☐ clear	SHIFT = CLR/HOME	8,24
⏮ left	SHIFT = ←CRSR→	7,9,10
◻ white	CTRL = 2	8
◆ Diamond	SHIFT = 2	9
⏏ f1	f1	11
⏏ f3	f3	12
⏏ f5	f5	13

```

1 REM PROGRAM 28
2 REM LO-RES SCREEN SAVE BY R. ALONSO
3 REM MODIFIED BY J. ANDERSON
4 REM-----
5 POKE 650,128
6 P=PEEK(197):POKE 53280,0:POKE 53281,0
7 SC=1024:CR=55296:BC=53280:C$=""
8 INPUT"NAME OF DRAWING";A$:PRINT"
9 GET B$:IF B$=""OR B$=CHR$(34) THEN B$=
"+!"
10 IF P=2 OR P=7 OR P=1 THEN C$="" :PRI
NT C$;
11 IF B$=" " THEN PRINT C$:GOTO 15
12 IF B$=" " THEN PRINT C$:GOTO 19
13 IF B$="!" THEN PRINT C$:END
14 PRINT B$;:GOTO 9
15 FOR X=1 TO 15:POKE BC,X:NEXT
16 OPEN 2,8,2,"0:"+A$+",",S,W"
17 FOR X=0 TO 999:S=PEEK(SC+X):C=PEEK(CR
+X)
18 PRINT#2,S:PRINT#2,C:NEXT:CLOSE 2:GOTO
9
19 FOR X=1 TO 15:POKE BC,X:NEXT
20 OPEN 2,8,2,"0:"+A$+",",S,R"
21 FOR X=0 TO 999:INPUT#2,S,C:POKE SC+X,
S
22 POKE CR+X,C:NEXT:CLOSE 2:GOTO 9
23 OPEN 15,8,15:INPUT#15,W$,X$,Y$,Z$
24 PRINT" ",W$,X$,Y$,Z$
25 CLOSE 15

```

When you run the program, it will ask you to name the drawing you want to design or retrieve. After you have input a name, you may go ahead and design a screen. Use the cursor keys as you would from the editing mode to move the cursor. You can use all the colors you want, RVS mode, text, and graphics characters. When you are ready to save your masterpiece, press f1.

To retrieve a screen that you have saved previously, enter the name of the screen you want at the opening prompt of the program, then press f3. The screen will load.

Using this program simplifies low-res graphics to just about their utter simplest. Now you can design low-res screens to your heart's content, without losing them when the computer is turned off.

■■■■■■■■■■

## SOUND FROM BASIC

We'll make no bones about it—programming sound from plain old Basic is no easy chore. In fact, we won't even examine it in detail here. Rather than trying to explain the fundamentals of Commodore sound at this point, we will supply some prepackaged examples that you can use in your own programs.

Feel free to experiment with the programs that follow, and make sure that you read the section on sound from Simon's Basic up ahead. Though it still requires effort, at least there are special sound commands at your disposal

in Simon's Basic that simplify sound with the C-64 a great deal.

But the programs that follow don't sound bad, and simply by playing with them you can change them and make them your own originals.

---

```

1 REM PROGRAM 29
2 REM SOUND EFFECTS FROM BASIC
3 REM MACHINE GUN
4 REM-----
5 S=54272
10 FORL=0TO24:POKES+L,0:NEXT
30 POKES+0,140:POKES+1,31
40 POKES+5,38:POKES+20,104
50 POKES+21,23:POKES+24,12
80 FORN=1TO15:POKES+4,129
100 FORT=1TO60:NEXT:POKES+4,128
110 FORT=1TO20:NEXTT,N:POKES+24,0

```

---



---

```

1 REM PROGRAM 30
2 REM SOUND EFFECTS FROM BASIC
3 REM WIND CHIME
4 REM-----
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
25 POKES+4,18
30 POKES+1,10
35 POKES+6,7
40 POKES+5,12
50 POKES+15,2
60 POKES+24,143
70 POKES+4,19
80 GOTO25

```

---



---

```

1 REM PROGRAM 31
2 REM SOUND EFFECTS FROM BASIC
3 REM RED ALERT ALARM
4 REM-----
5 S=54272
10 FORL=0TO24:POKES+L,0:NEXT
30 POKES+0,23:POKES+1,15
40 POKES+5,155:POKES+20,200
50 POKES+21,255:POKES+24,89
80 POKES+4,99
85 FORX=1TO250:NEXT
90 POKES+4,0:GOTO30

```

---

---

```

1 REM PROGRAM 32
2 REM SOUND EFFECTS FROM BASIC
3 REM SAWING WOOD
4 REM-----
5 S=54272
10 FORL=0TO24:POKES+L,0:NEXT
30 POKES+0,23:POKES+1,15
40 POKES+5,155:POKES+20,200
50 POKES+21,255:POKES+24,89
80 POKES+4,129
85 FORX=1TO300:NEXT
90 POKES+4,0:GOTO30

```

---



---

```

1 REM PROGRAM 33
2 REM SOUND EFFECTS FROM BASIC
3 REM LASER GUN
4 REM-----
5 S=54272
10 FORL=0TO24:POKES+L,0:NEXT
30 POKES+0,23:POKES+2,155
40 POKES+5,155:POKES+20,200
50 POKES+21,255:POKES+24,89
80 POKES+4,99
85 FORX=1TO255:POKES+1,X:NEXT
90 POKES+4,0:GOTO30

```

---



---

```

1 REM PROGRAM 34
2 REM SOUND EFFECTS FROM BASIC
3 REM BONUS POINTS
4 REM-----
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
25 POKES+4,31:POKES+5,25
35 POKES+6,27:POKES+7,12:POKES+24,15
75 FORX=1TO15:POKES+15,25-X:FORDE=1TO25:
POKES+15,X+25:NEXTDE,X
80 GOTO25

```

---



---

```

1 REM PROGRAM 35
2 REM SOUND EFFECTS FROM BASIC
3 REM BOMBS AWAY
4 REM-----
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
25 POKES+4,29:POKES+5,23
30 POKES+1,16:POKES+2,16

```

---

```

35 POKES+6,27:POKES+7,12:POKES+24,15
75 FORX=1TO150:POKES+15,255-X:NEXT
80 GOTO25

```

---

```

1 REM PROGRAM 36
2 REM SOUND EFFECTS FROM BASIC
3 REM ELECTRONS
4 REM-----
5 S=54272
10 FORL=0TO24:POKES+L,0:NEXT
30 POKES+0,23:POKES+1,15
40 POKES+5,155:POKES+20,200
50 POKES+21,255:POKES+24,89
80 POKES+4,99
85 FORX=1TO25:POKES+0,X:POKES+1,255-X
90 FORDE=1TO50:NEXTDE,X
95 POKES+4,0:GOTO30

```

---

```

1 REM PROGRAM 37
2 REM SOUND EFFECTS FROM BASIC
3 REM OUT OF FUEL
4 REM-----
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
25 POKES+4,18
30 POKES+1,10
35 POKES+6,27
40 POKES+5,12
50 POKES+15,2
70 POKES+4,29
75 FORX=1TO150:POKES+24,X:POKES+15,X:NEXT
80 GOTO25

```

---

```

1 REM PROGRAM 38
2 REM SOUND EFFECTS FROM BASIC
3 REM MARTIAN POGO STICK
4 REM-----
5 S=54272
10 FORL=0TO135:POKES+L,0:NEXT
30 POKES+0,22:POKES+1,12
40 POKES+5,135:POKES+20,193
50 POKES+21,255:POKES+24,89

```

```
80 POKES+4,99
85 FORX=1TO 5:POKES+0,X:POKES+1,255-X
90 FORDE=1TO50:NEXTDE,X
95 POKES+4,0:GOTO30
```

---

```
1 REM PROGRAM 39
2 REM SOUND EFFECTS FROM BASIC
3 REM SPACE MACHINERY
4 REM-----
5 S=54272
10 FORL=0TO24:POKES+L,0:NEXT
30 POKES+0,23:POKES+2,150
40 POKES+5,10:POKES+20,255
50 POKES+61,10:POKES+24,100
80 POKES+4,83
85 FORX=1TO 25:POKES+1,X:NEXT
90 POKES+4,0:GOTO30
```

---



## GETTING INTO SIMON'S BASIC

---

Loading Simon's Basic is as simple as plugging in the cartridge in the back of the Commodore 64, then turning the computer on. Make sure that the label side faces up, and don't plug the Simon's Basic cartridge into or out of the computer while it is turned on. This can cause damage to the cartridge and to the computer.

Other than that, there isn't much more you can do seriously wrong. Plug it in and start cooking. Simon's Basic adds 114 new commands to Basic, many of them related to graphics and sound control.

When Simon's Basic boots up, the screen will be white with a blue background. The characters will be black, and easy to read. You should see the words "Extended CBM V2 Basic", "30719 Basic Bytes Free", followed by a "Ready" prompt. This indicates that all is well, and you are indeed ready to begin programming in Simon's Basic.

Disk commands to save and load are the same as in plain old Basic. If you need a refresher, refer back to the chapter on the disk drive.

### **SIMON'S BASIC COMMANDS**

There are, however, a few new commands you should know about. You can now obtain a disk directory at any time simply by typing `DIR$` `[RETURN]`. This will not affect any program currently in memory. In addition, Simon's Basic allows simplified disk-handling using the command `DISK`. The following table shows how to execute disk commands directly from Simon's Basic. Alternatively you may still use the `MENU` program, presented earlier in the book. It runs perfectly well in Simon's Basic, and is still easier to use than Simon's Basic `DISK` commands.

Simon's Basic disk commands.

```

DIR "$" ————— OBTAIN DISK DIRECTORY
DISK "NØ:DISK NAME,01" ———— FORMAT A DISK
DISK "SØ:FILE NAME,01" ———— SCRATCH A FILE
DISK "CØ:NEWFILE=0:OLDFILE" ——— COPY A FILE
DISK "RØ:NEWNAME=OLDNAME" ——— RENAME A FILE

```

↑  
Note these are zeros, not "ohs"

Now let's start learning about Simon's Basic graphic and programming commands.

## COLOUR

Want to change screen background and border colors? Simple as can be. Use the command COLOUR.

That spelling is correct for the British, and it was incorporated as such into Simon's Basic. So whenever you want to say the word "color" to Simon's Basic you will have to spell it the British way. Here's the way it might look as a direct command:

```
COLOUR 4,7 RETURN
```





This command will turn the center of the screen yellow and the border purple. If you read the section on plain old Basic, you will know how much simpler the COLOUR command makes it to switch screen and background colors. Type COLOUR, then the border color number you want, then the screen color number you want. Separate the two numbers with a comma. The format is

COLOUR border,screen

Watch out, early owners of Simon's Basic. The manual gets it wrong here (as it does in several places). The order of commands is COLOUR border,screen—not the other way around.

The color codes are just the same for Simon's Basic as they are for old-fashioned Basic. Here they are again:

Simon's Basic color codes.

Black ————— 0	Orange ————— 8
White ————— 1	Brown ————— 9
Red ————— 2	Pink ————— 10
Medium Blue ——— 3	Dark Grey ————— 11
Purple ————— 4	Medium Grey ——— 12
Green ————— 5	Light Green ——— 13
Blue ————— 6	Light Blue ——— 14
Yellow ————— 7	Light Grey ——— 15

Look how easy it is to write a color flashing program in Simon's Basic:

```

1 REM PROGRAM 40
2 REM FLASHING COLORS FROM
3 REM SIMON'S BASIC
4 REM-----
10 FOR X=0 TO 15
20 COLOUR X, X+1
30 FOR Y=1 TO 300:NEXT Y
40 NEXT X
50 GOTO 10

```

Perhaps it is becoming clear that we are now dealing with a language that will finally allow us to get a reasonable result for the amount of work we put in!

■■■■■■■■■■

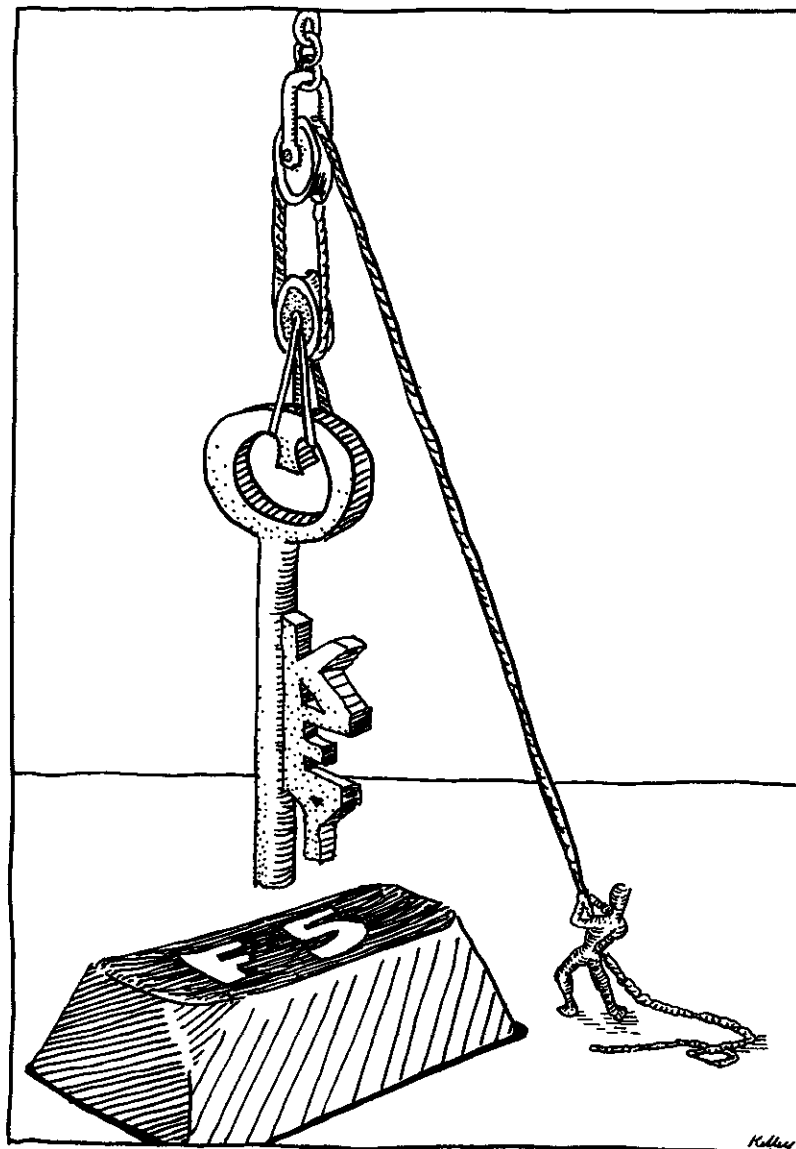
**KEY**

Before we go on to more advanced graphics, let's look at a few other facets of Simon's Basic that will make all of our programming much easier.

The command KEY allows you to define a string or command as a simple press of the function key. This comes in really handy with functions that are repeated over and over—instead of having to retype them each time you need them, you can make them KEY commands.

For example, let's say you are writing a program that uses many COLOUR statements throughout the listing. Instead of typing the word COLOUR over and over you could type the following just once:

```
KEY 5 "COLOUR" RETURN
```



Now, whenever you need to type the command COLOUR, all you need to do is hit the f5 key (for "function key 5"), on the lower righthand side of the C-64. This is called "macro definition" on more sophisticated systems and it is a very powerful capability. The format for the command is

KEY n "string"

where:

n = 1 - 16

string = words, graphics characters, commands to define as key

You may even embed low-res color graphics statements into KEY macros. This can speed screen development for the screen save program shown in the previous section. Here is an example:

---

```

1 REM PROGRAM 41
2 REM EMBEDDING LOW-RES COLOR GRAPHICS
3 REM INTO SIMON'S BASIC
4 REM-----
10 KEY 1, "EO"
20 KEY 3, "E"
30 KEY 5, "E"
40 KEY 7, "E"
50 DISPLAY

```

---

Note that it is a good idea to return the cursor color to default at the end of the macro, so you will readily be able to read whatever you type next. Actually, you can leave the cursor any color you want, as long as you make sure it is compatible with the background color. Otherwise you won't be able to see what you're typing.

Remember that whenever that does happen, you can always hit the RUN/STOP and RESTORE buttons together to return to the default color combination.

Let's look at another very powerful potential of the key command. Type in the following:

```
KEY 1, "LIST"+CHR$(13) RETURN
```

By typing +CHR\$(13), you have taught the computer how to "hit RETURN" on its own. Now, whenever you want to list the program currently in memory, all you need to do is hit f1 (for function key 1), at the top righthand side of the keyboard. To make LISTing and RUNning the programs a snap when working with Simon's Basic, keep f1 defined as the LIST command, and F7 as the RUN command. You can do this by typing

```
KEY 7, "RUN"+CHR$(13) RETURN
```

There are 16 possible combinations of function keys and 16 possible KEY macros, each with a maximum length of 15 characters. The chart below shows how each function key is pressed:

How to type key command functions.

f1	<b>f1</b>	f9	<b>⇧—f1</b>
f2	<b>SHIFT—f1</b>	f10	<b>⇧—f2</b>
f3	<b>f3</b>	f11	<b>⇧—f3</b>
f4	<b>SHIFT—f3</b>	f12	<b>⇧—f4</b>
f5	<b>f5</b>	f13	<b>SHIFT—⇧—f1</b>
f6	<b>SHIFT—f5</b>	f14	<b>SHIFT—⇧—f2</b>
f7	<b>f7</b>	f15	<b>SHIFT—⇧—f3</b>
f8	<b>SHIFT—f7</b>	f16	<b>SHIFT—⇧—f4</b>

You can get around the length restriction by devoting two macros to a single string. For instance, you might type

```
KEY 3, "COMMODORE 64 GR" RETURN
KEY 5, "APHICS & SOUND" RETURN
```

Then you would use the two keys together to achieve the complete macro.

\*\*\*\*\*

## DISPLAY

In order to see a full list of the defined function keys, type **DISPLAY RETURN**. All function key commands will be listed.

You can use the KEY and DISPLAY commands in the direct or in the deferred mode, as well. You can load the program below, for example, and automatically load your own custom set of macro definitions. RUN the program, and they will be installed. You can then load in new programs, without disturbing defined KEY functions.

```
1 REM PROGRAM 42
2 REM KEYING KEYBOARD FUNCTIONS
3 REM INTO SIMON'S BASIC
4 REM-----
```

```

10 KEY 1, "LIST"+CHR$(13)
20 KEY 3, "LOAD"
30 KEY 5, "SAVE"
40 KEY 7, "RUN"+CHR$(13)
50 KEY 2, "PRINT"
60 KEY 4, "FOR"
70 KEY 6, "NEXT"
80 KEY 8, "REM"
90 DISPLAY

```

## AUTO

You may also define key functions to be accessed from within a program or define multiple custom macro sets. As you can see, the KEY function is a major convenience of Simon's Basic.

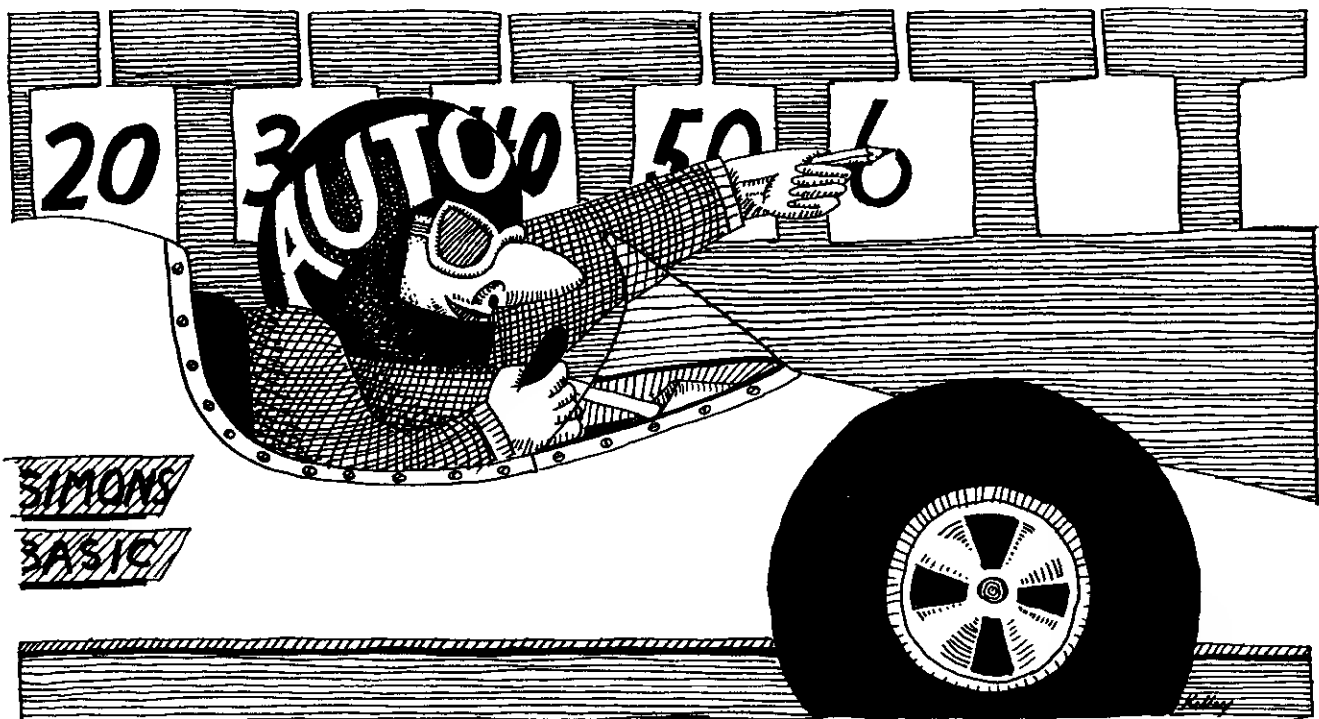
When programming in Simon's Basic, you can use the AUTO command to generate line numbers automatically. The format is

AUTO line number, increment

where:

line number = legal Basic starting line number

increment = amount to count



The parameter line number is where you want to begin the code, and increment is the amount to skip between numbers. The command

```
AUTO 10,10 RETURN
```

will begin automatic line numbering at 10, and continue with 20, 30, 40 and so on. Every time you press return, a new line number will be generated and displayed at the start of the new line.

To end an auto function, simply press `RETURN` without typing anything else on that line. This will return you to default operation.

RENUMBER

## RENUMBER

Another line numbering capability of Simon's Basic is the function RENUMBER. It works just the way AUTO does, only it is used on an existing sequence of code. The command RENUMBER allows you to move a chunk of code around within a program without having to retype it. It will also allow you to recover space between two consecutive line numbers. If you have a line 12, a line 13, and you need something in between, it is time to use RENUMBER.

The format for RENUMBER matches AUTO exactly:

RENUMBER line number, increment

It is very important to remember one thing when you are using the RENUMBER command. It does *not* renumber any referenced line numbers—that is to say, it will not keep track of changes in GOTO or GOSUB commands. Later on in Simon's Basic, we will build procedures and learn how they can be used instead of GOTO and GOSUB commands, which are hard to keep track of. But if you use them, then use RENUMBER, you will have to change the referenced lines "by hand."

PAUSE

## PAUSE

Remember how we used a FOR-NEXT loop to cause a time delay in earlier Basic graphics programs? Well Simon's Basic allows us to create pauses in a much simpler way. Simply type

```
PAUSE 5 RETURN
```

within the body of a program, and it will pause for 5 seconds. We shall be using PAUSE quite a bit up ahead, so get used to it.

You can also insert a PRINT statement directly into a PAUSE command. The format is as follows:

```
PAUSE "MORE IN TEN SECONDS",10 RETURN
```

The PAUSE command saves us the trouble of adding a PRINT statement. It lets us insert a message within the framework of the pause command itself.

So the format for PAUSE is

PAUSE "optional string", time

where:

optional string = message (may be omitted)

time = seconds for pause

■■■■■■■■■■

## ON ERROR: GOTO

One of the major oversights of the original Basic that ships in the C-64 is its lack of error-trapping. This means that when a situation arises where an error is detected, the program breaks, execution ceases, and the jig is up. Once a program hits an error, it crashes. The ON ERROR:GOTO command allows you to trap program errors—to actually use them to your advantage if you care to.

ON ERROR:GOTO allows you to branch to a specified line of code whenever an error is detected. The format for the command is

ON ERROR:GOTO line number

where:

line number = line to branch to when error is detected

So what's the big deal about trapping errors? Well in this book we'll use the command to keep screen graphics on the screen. You will think up your own uses for the command before too long. Here's a simple example:

---

```

1 REM PROGRAM 43
2 REM ON ERROR EXAMPLE
3 REM USE RUN/STOP TO END
4 REM-----
10 X=0:ON ERROR:GOTO 100
20 POKE 53280,X:POKE 53281,X+1:PRINT X
30 X=X+1:GOTO 20
100 PRINT"WHEN X>254 THE ERROR IS TRAPPE
D. "

```

---

■■■■■■■■■■

## OUT

The OUT command acts to disable the current ON ERROR:GOTO statement—so you may continue with program execution. After you branch to the line specified by ON ERROR:GOTO, insert an OUT command to clear the error code. Then when you head back into code that includes the original or a

new ON ERROR command, it won't be triggered by an old error, which has already been taken care of.

Here is another example of ON ERROR, this time reset by the command OUT:

---

```

1 REM PROGRAM 44
2 REM ON ERROR EXAMPLE
3 REM WITH NO ERROR RESET
4 REM-----
10 X=0:ON ERROR:GOTO 100
20 POKE 53280,X:POKE 53281,X+1:PRINT X
30 X=X+1:GOTO 20
100 PRINT"WHEN X>254 THE ERROR IS TRAPPE
D. "
110 NO ERROR:REM RESETS ERROR MODE
120 PRINT"GOING BACK TO LOOP AGAIN. "
130 FOR Y=1 TO 999:NEXT Y
140 GOTO 10

```

---

\*\*\*\*\*

## SIMON MEETS LOW-RES GRAPHICS

We've taken an excursion through some commands that make programming in Simon's Basic much easier than in any other version of Basic, for the Commodore or any other comparable machine. Now let's take a look at what we can do with low-resolution graphics from Simon's Basic.

\*\*\*\*\*

## SCRV and SCRLD

Perhaps the most important low-res commands available from Simon's Basic are SCRSV and SCRLD. SCRSV stands for "screen save," which allows you to save a low-res screen. SCRLD, which stands for "screen load," allows you to load a screen saved with a SCRSV command.

Instead of having to go through the trouble of keying in a program to load and save low-res screens, as we did from plain old Basic, Simon's Basic has dedicated commands to make the job simple for us. They can be used in the direct or the deferred modes.

To save a screen to disk, you use the following command format:

SCRSV 2,8,2,"screen filename,S,W"

where:

screen filename = name you have chosen for screen

S,W = never change, enter as shown

The parameters other than the screen filename, which you will supply, will always remain the same—these tell Simon's Basic to open a file to disk, write screen data to it sequentially, and then mark it with an end-of-file marker.

To load a screen that has been saved to disk, follow this format:



SCRLED 2,8,2,"screen filename"

It's just that simple. The screen you saved as that filename will appear on the screen. Any low-res screens, even those in color and using special graphics characters, can be saved in this manner. In this way you can preserve your low-res works of art forever.

■■■■■■■■■■

## BCKGNDS

The BCKGNDS command stands for "backgrounds," and allows you to change the background color of a category of characters. BCKGNDS is used to make text look bolder and more appealing. You may not get a feeling for how neat multicolored BCKGNDED text looks until you give this command a try.

One thing you should bear in mind is that the BCKGNDS command cannot be used with graphics characters—only with text characters. So this command is only used to enhance the looks of a specific letter, word, or sentence.

The format for the command is

BCKGNDS screen color,background color 1,background color 2,background color 3

where:

screen color = 0 – 15

background color 1 = 0 – 15

background color 2 = 0 – 15

background color 3 = 0 – 15

The parameter screen color defines the color of the background screen itself. The next parameter, background color 1, determines what background color is assigned to all SHIFTed characters. The parameter background color 2 determines the background color assigned to all REVERSE-FIELD unSHIFTed characters. The final background parameter determines the background color of REVERSE-FIELD SHIFTed characters.

Here is a good example of the BCKGNDS command at work in a text program.

---

```

1 REM PROGRAM 45
2 REM BCKGNDS ALLOWS FOR COLOR
3 REM TEXT AND LOW-RES BACKGROUNDS
4 REM-----
10 BCKGNDS 0,2,6,4
20 PRINT:PRINT:PRINT:PRINT
30 PRINT "NOW EXPERIMENT TYPING WITH
  THE SHIFT"
40 PRINT"KEY HELD DOWN, IN AND OUT OF
  RVS MODE. "
50 STOP

```

---

\*\*\*\*\*

**FLASH and OFF**

Using the FLASH command is another good way to draw attention to the screen itself or to certain words on the screen. It can take either of two alternate formats:

FLASH color,speed

where:

color = 0 — 15

speed = 1 — 255

This command enables you to flash all characters in a single color, at a rate of speed that is selectable. The speed range may be any number from 1 to 255—with 1 as the fastest flash rate and 255 the slowest.

Alternatively, you may use this format:

FLASH color

where:

color = 0 — 15

This will flash a color at a default rate of every 4 seconds.

To stop flashing when you are ready to do so, simply use the command OFF. The only trick to using the OFF command is to make sure to turn off the flashing at a moment when the characters are visible—otherwise you could run into problems with an invisible cursor.

As a remedy to off-timing FLASH, you could redefine cursor color after using an OFF command.

Here are a couple of uses of FLASH:

---

```

1 REM PROGRAM 46
2 REM USING "FLASH"
3 REM
4 REM-----
10 PRINT:PRINT:PRINT:PRINT
20 PRINT "BYOUR ATTENTION■, PLEASE."
30 FLASH 1, 50

```

---



---

```

1 REM PROGRAM 47
2 REM USING "FLASH"
3 REM IN MULTIPLE COLORS
4 REM-----

```

---

---

```

10 PRINT:PRINT:PRINT:PRINT
20 PRINT "          PLEASE!"
30 FLASH 7, 10:FLASH 2,10
40 PRINT "DON'T TOUCH THAT DIAL."

```

---



---

## BFLASH

If you really want to get flashy, you can resort to the BFLASH command. This stands for "border flash," and flashes the screen border color at a rate you may stipulate. The format is as follows:

BFLASH speed, color 1,color 2

where:

speed = 1 – 255

color 1 = 0 – 15

color 2 = 0 – 15

The first parameter determines the rate of the flash, and the following parameters determine which colors shall flash. To turn the flashing off, use the command BFLASH 0 and everything will revert to normal once again. Here's a simple command trial:

---

```

1 REM PROGRAM 48
2 REM USING "BFLASH"
3 REM
4 REM-----
10 PRINT:PRINT:PRINT:PRINT
20 PRINT "A REAL ATTENTION-GETTER!"
30 BFLASH 30, 4, 7

```

---



---

## FILL

The command FILL allows you to fill a rectangular area of the screen (in a size and location you choose) with text characters of a specific color and type. Command format is as follows:

FILL row,column,width,length,character code,color

where:

row = 0 – 24

column = 0 – 39

width= 1 – 24

length= 1 — 39  
 character code= POKE code for selected text character  
 color= 0 — 15

The first four parameters in the FILL command define the area of the screen to be FILLED. Rows are numbered 0 to 24, and columns from 0 to 39. The parameters row and column represent the top lefthand point of the FILL, and the parameters width and depth represent the size of the rectangular shape to be filled. The next parameter, character code, is the POKE code associated with the character that will comprise the fill.

The final parameter is the color of the FILL character you desire. Here is one form that use of the FILL command might take:

---

```
1 REM PROGRAM 49
2 REM THE "FILL" COMMAND
3 REM AN EASY EXAMPLE
4 REM-----
10 FILL 5,3,15,17,65,4
```

---

## MOVE

MOVE is an extremely powerful and useful command that allows you to copy a section of the screen and move it elsewhere on the screen. The command format is as follows:

MOVE row,column,width,length,destination row,destination column

where:

row = 0 — 24  
 column = 0 — 39  
 width = 1 — 24  
 length = 1 — 39  
 destination row = 0 — 24  
 destination column = 0 — 39

The first four command parameters define the screen area you wish to reproduce, starting with the upper lefthand corner, and defining the size of the block. The last two parameters specify the row and column coordinates of the top lefthand corner of the area where the screen will be duplicated.

Be sure that the parameters you use do not result in a MOVE exceeding the limits of the screen. This means that the depth of the screen area to be duplicated added to the row number of the area into which the information is to be reproduced must not exceed 25. It also means that the column number of the area into which the data is to be reproduced must not be greater than 40. No MOVE command can execute if those parameters result in a BAD MODE error.

Here are some examples of MOVE commands that work—at work:

---

```

1 REM PROGRAM 50
2 REM THE "MOVE" COMMAND
3 REM
4 REM-----
10 PRINT"  □  □  □  □  □  "
20 PRINT"  □  □  □  □  □  "
30 PRINT"  □  □  □  □  □  "
40 PRINT"  □  □  □  □  □  "
50 MOVE 0,0,5,5,10,10

```

---



---

```

1 REM PROGRAM 51
2 REM THE "MOVE" COMMAND
3 REM ANOTHER EXAMPLE
4 REM-----
10 PRINT"  □  □  □  □  □  "
20 PRINT"  □  □  □  □  □  "
30 PRINT"  □  □  □  □  □  "
40 PRINT"  □  □  □  □  □  "
50 MOVE 0,0,5,5,5,5
60 MOVE 5,5,5,5,10,10
70 MOVE 10,10,5,5,15,15

```

---



---

```

1 REM PROGRAM 52
2 REM THE "MOVE" COMMAND
3 REM YET ANOTHER EXAMPLE
4 REM-----
10 PRINT"  □  □  □  □  □  "
20 PRINT"  □  □  □  □  □  "
30 PRINT"  □  □  □  □  □  "
40 PRINT"  □  □  □  □  □  "
50 MOVE 0,0,5,5,5,5
60 MOVE 5,5,5,5,10,10
70 MOVE 10,10,5,5,15,15
80 MOVE 0,0,20,20,5,20

```

---



---

```

1 REM PROGRAM 53
2 REM THE "MOVE" COMMAND
3 REM MOVING PART OF THE ORIGINAL
4 REM-----
10 PRINT"  □  □  □  □  □  "
20 PRINT"  □  □  □  □  □  "

```

---

```

30 PRINT" HHH"
40 PRINT" LLLL"
50 MOVE 0,0,3,3,10,10

```

---

```

1 REM PROGRAM 54
2 REM THE "MOVE" COMMAND
3 REM MOVING FOR ARTFUL EFFECT
4 REM-----
10 PRINT"  "
20 PRINT"  "
30 PRINT"  "
40 PRINT"  "
50 PRINT"  "
60 PRINT"  "
70 PRINT"  "
80 PRINT"  "
90 PRINT"  "
100 PRINT"  "
110 PRINT"  "
120 PRINT"  "
130 MOVE 0,0,15,15,10,20
140 MOVE 0,0,5,5,7,7
150 MOVE 5,5,9,9,10,10
160 MOVE 10,10,5,5,0,0

```

---

## SCROLL

Simon's Basic provides a command to enable you to scroll (smoothly move) specified areas of screen data in any one of four directions: LEFT, RIGHT, UP, or DOWN. The power of the SCROLL command is formidable, and makes low-res animation easy. The format for the SCROLL command is as follows:

direction,scroll type,start row,start column,end column,end row

where:

direction = LEFT, RIGHT, UP, or DOWN

scroll type = W or B

start row = 0 - 24

start column = 0 - 39

end column = 0 - 39

end row = 0 - 24

The first parameter in the scrolling command specifies the direction in which scrolling will take place. The second command parameter is either a W or a B, to indicate "wrap-around" or "blinking." If a section of the screen is

scrolled with wrap-around, any characters within the specified screen area will scroll off the edge of the area only to reappear at the opposite edge. When you use blanking, data that scrolls off-screen will not reappear.

The parameters start row and start column in a scrolling command define the row and column coordinates at the start of the area you wish to scroll.

Likewise, parameters end column and end row specify the column and row coordinates of the end of the scroll area.

Scrolling commands may be combined in order to scroll different areas of the screen in different directions simultaneously. But bear in mind: the maximum height and width of any scroll area cannot exceed 24 lines down or 23 characters across.

Bet you'd like to see some examples of SCROLL at work in programs. Here you go:

---

```

1 REM PROGRAM 55
2 REM THE "SCROLL" COMMAND
3 REM
4 REM-----
10 PRINT"  HHHH"
20 PRINT" HHHH"
30 PRINT" HHHH"
40 PRINT" HHHH"
50 RIGHTW 0,0,39,5
60 GOTO 50

```

---



---

```

1 REM PROGRAM 56
2 REM THE "SCROLL" COMMAND
3 REM IN TWO DIRECTIONS AT ONCE
4 REM-----
10 PRINT"  HHHH"
20 PRINT" HHHH"
30 PRINT" HHHH"
40 PRINT" HHHH"
50 RIGHTW 0,0,40,5
60 DOWNW 0,0,9,24
70 GOTO 50

```

---



---

```

1 REM PROGRAM 57
2 REM THE "SCROLL" COMMAND
3 REM TWO SHAPES IN TWO DIRECTIONS
4 REM-----
10 PRINT"  HHHH"          *****"
20 PRINT" HHHH"          *****"
30 PRINT" HHHH"          *****"

```

---

```
40 PRINT "      " *****  
50 LEFTW 0,0,40,5  
60 DOWNW 0,13,5,24  
70 GOTO 50
```

---

As you can see, low-res graphics need not be a second-class operation—in some ways, the graphics potentials of low-res outclass even the so-called “graphics” screens we are about to discover.

■■■■■■■■■■

## HI-RES GRAPHICS

Simon's Basic puts the powers of high-resolution graphics at the fingertips of the beginning programmer. With just a few simple commands, you can build complex, multicolor pictures. Let's get right to it.





## HIRES

So far we have done all our graphics from the low-resolution or text mode. You can do a lot more than just text from the text mode, as we hope you saw in the previous section. But the high-resolution mode allows you to do even more. The HIRES command tells the computer to go into high-resolution mode, and in what screen and plot color to do so. By plot color, we mean the color the C-64 will use to draw on the hi-res background color when we get around to drawing on it. The format is

HIRES screen color, plot color **RETURN**

So if you type HIRES 0,7 **RETURN** you should see a hi-res screen in yellow, with black as the plot color, right? Try it.

What went wrong? There was just a flash, and then the text screen returned. The reason for this is that unless we are in the process of doing something on a hi-res screen, the C-64 will automatically return us to the text mode. We can get around this by telling the computer not to come back.

---

```

1 REM PROGRAM 58
2 REM THE "HIRES" COMMAND
3 REM
4 REM-----
10 HIRES 0,7
20 GOTO 20

```

---

Here we wrote a program line to keep us in the hi-res mode. Big deal. Looks like a plain yellow screen to me. But now let's learn how to draw shapes on the screen without using cumbersome character graphics.

## LINE

The LINE command allows us to draw a line on the screen from one point to another. The following program draws a diagonal line across the screen:

```

10 HIRES 0,1
20 LINE 20,40,300,250,1
30 GOTO 30

```

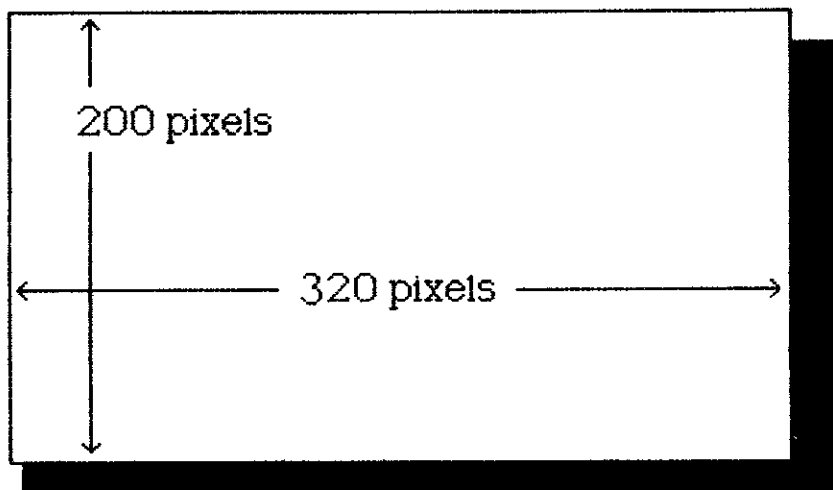
The format for the LINE command is as follows:

LINE beg x, beg y, fin x, fin y, plot type

where beg x is the beginning x value of the line, beg y is the beginning y value, fin x is the final x value, fin y is the final y value, and the plot type is set to 0, 1 or 2. For now we will always use a plot type set to 1.

At first you are bound to be confused by the placement of x and y plots across the screen. The chart below may help give you a feel for the hi-res screen:

Dimensions of the hi-res screen.



Play around with the x and y values for line plots. Before long you'll get a feeling for the hi-res screen.

Now let's get a taste of the animation potential of Simon's Basic in hi-res. The next simple program allows you to create an animated line plot:

---

```

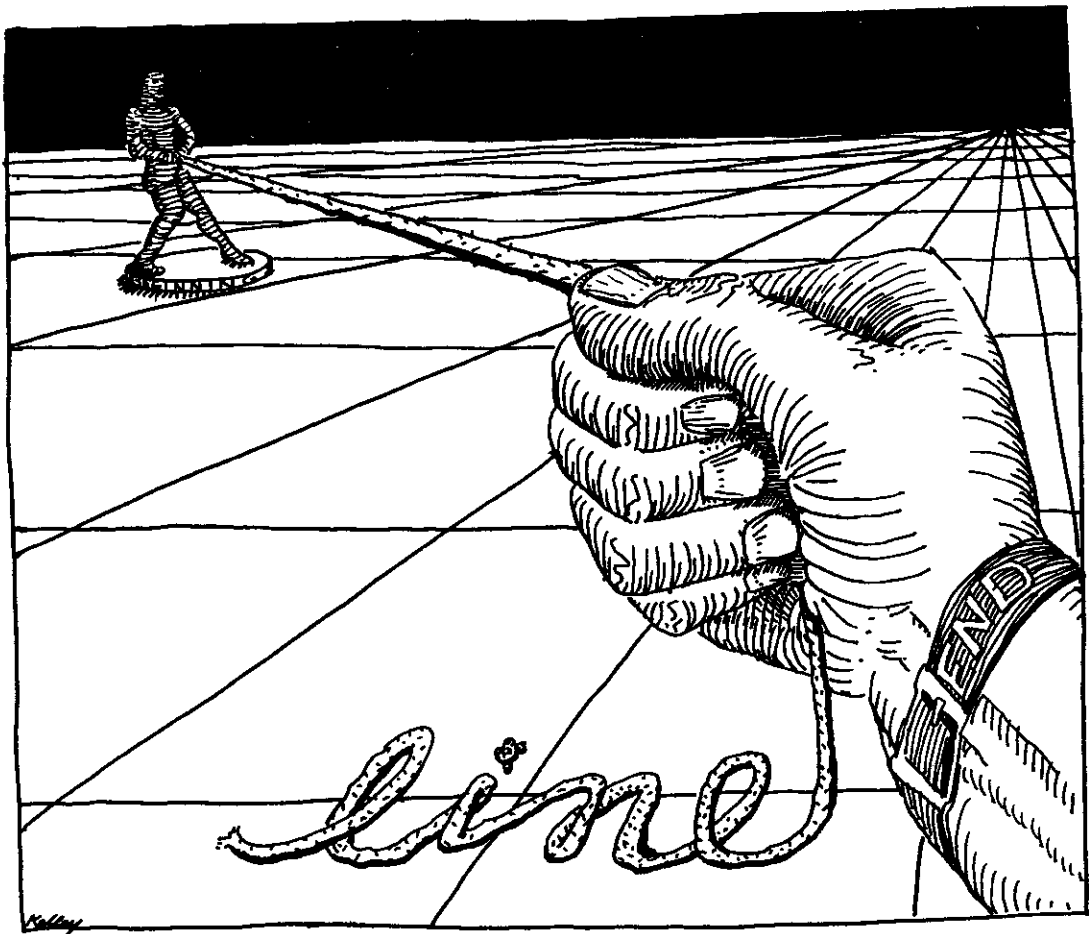
1 REM PROGRAM 59
2 REM ANIMATED LINE PLOT
3 REM
4 REM-----
10 HIRES 0,7:Y=0
20 LINE 0,0,300,Y,1
30 Y=Y+4:IF Y>250 THEN 50
40 GOTO 20
50 GOTO 50

```

---

We shall be building on this basic principle to create sophisticated moving pictures. It is used simply here, but can be used in sophisticated ways as well.

As you can see from this listing, all you need to do is build a counter for the x and y values, then set up a loop. Each time through the loop, the line plots to the incremented x and y values. The result: an animated drawing.



## REC

Any guess as to what the REC command allows us to draw on a hi-res screen? You got it, a rectangle. As with LINE, it takes a while to get used to the numbers you need to use with the rectangle command, but you can get the hang of it with a bit of practice.

First you tell the computer where you want the top lefthand corner of the rectangle to be. You do this by specifying X and Y coordinates. The only way to really get a feel for these coordinates is to experiment.

Next you tell the computer how wide and how long you want the rectangle to be. Again, it takes practice to see what numbers give you the result you want.

Let's try one:

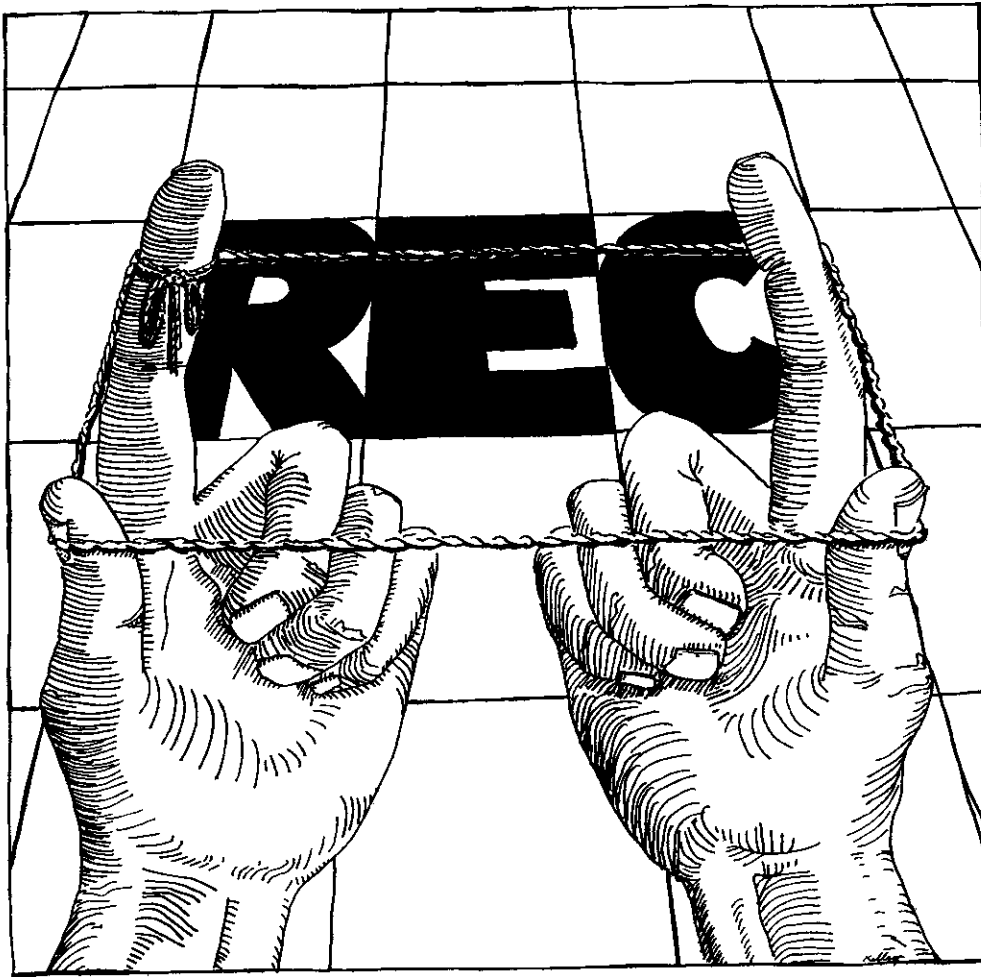
---

```

1 REM PROGRAM 60
2 REM THE REC COMMAND
3 REM
4 REM-----
10 HIRES 0,7
20 REC 50,50,120,120,1
30 GOTO 30

```

---



You can see where this set of coordinates places the rectangle. The command is

`REC beg x,beg y,width,length,plot type`

where beg x and beg y are the coordinates of the top lefthand corner of the rectangle, width is the horizontal width of the rectangle, and length is the vertical length of the rectangle.

What about that strange number 1 tacked on the end of the expression? Well, it tells the computer that this is a normal plot. We will look at inverse and clear plots just ahead. For now all you need to know about is plot type 1.

First, let's put the REC command to work for us in an animated plot.

---

```
1 REM PROGRAM 61
2 REM ANIMATED REC PLOT
3 REM
```

---

```

4 REM-----
5 X=1:Y=30:A=150:B=150
10 HIRES 0,7
20 REC X,Y,A,B,1
30 X=X+5:Y=Y+1:A=A-2:B=B-3
35 IF X>250 THEN 50
40 GOTO 20
50 GOTO 50

```

---

Pretty good result for just a wee bit of effort, wouldn't you say? This program is very easy to understand, once you get a grip on the REC command. It is a loop, just like the ones we have set up before, and each time through the loop it increments the corners of the rectangle. The top lefthand corner moves down and to the right, while the bottom righthand corner moves up and to the right. This gives us a three-dimensional effect. We stop plotting when X gets to 251, so we won't error out. We actually begin plotting the entire figure again, but you can't see it happen because it is plotting right over itself.

---

## A Closer Look at Plot Types

Let's gain an understanding of plot types by playing with those from our last example program. We'll change the plot type of the program by changing the last value in the REC statement.

```

20 REC X,Y,A,B,0

```

If we specify a plot type of 0, nothing gets plotted. If a plot type 0 encounters a plotted line, it will actually erase it. This comes in handy for "undrawing" animated shapes. The following program will provide an example.

---

```

1 REM PROGRAM 62
2 REM UNDRAWING TO ANIMATE
3 REM
4 REM-----
10 HIRES 0,7:X=5:Y=5
20 REC X,Y,X+10,X+10,1
30 FOR Z=1 TO 10:NEXT Z
40 REC X,Y,X+10,Y+10,0
50 X=X+1:Y=Y+1
60 IF X>90 THEN 10
70 GOTO 20

```

---

As we saw earlier, a plot type of 1 is a normal plot, drawing a shape in the plot color across the background.

A plot type of 2 “inverses” whatever it encounters. It turns a plot off if it is on, and on if it is off. We can use a plot type of 2 to make our animated rectangle change shape continuously:

---

```

1 REM PROGRAM 63
2 REM REC PLOT UNDraws ITSELF
3 REM
4 REM-----
5 HIRES 0,7
10 X=1:Y=30:A=150:B=150
20 REC X,Y,A,B,2
30 X=X+5:Y=Y+1:A=A-2:B=B-3
35 IF X>250 THEN 10
40 GOTO 20

```

---

We shall learn about more plot types after we have learned about the multicolor mode. In the hi-res mode, 0, 1, and 2 are all the available plot types. Play with them until you get a feel for what they do.

■■■■■■■■■■

## MULTI-RES— THE BEST OF BOTH WORLDS

So far we have been looking at the hi-res mode, where we can plot one plot color on top of one background color. Now get ready for the multi-res mode, where we can plot in three plotting colors and, with a little sneaky footwork, even more.

The multicolor mode, which we shall call multi-res, is a variant of the hi-res mode—with half the horizontal resolution, but three times the color. It is up to you to decide which resolution to use with which graphics trade-offs.

■■■■■■■■■■

## MULTI

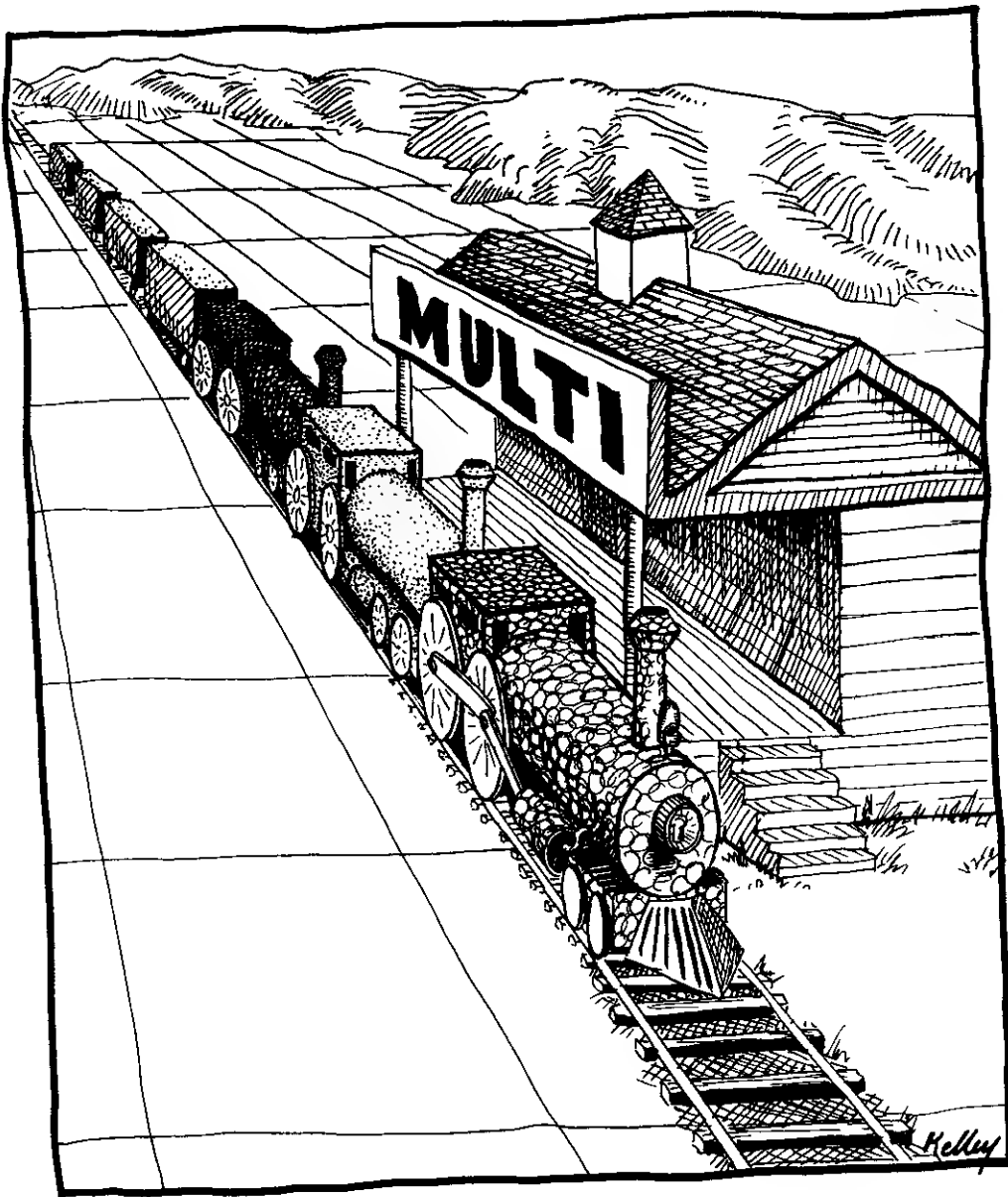
The MULTI command, when used following a call to the HIRES command, will cause all plotting to take place in multi-res. The format for the command is

HIRES plot color,background color:MULTI color 1,color 2,color 3

where:

plot color = 0 — 15  
background color = 0 — 15  
color 1 = 0 — 15  
color 2 = 0 — 15  
color 3 = 0 — 15

Note that a MULTI command must always follow a HIRES command. The three parameters following MULTI define the plot colors you wish to use.



Each plot color is selected by its MULTI command designation as the plot type in a plotting command. We will clarify this just ahead. But here are some examples of hi-res graphics transposed into multi-res:

---

```

1 REM PROGRAM 64
2 REM ANIMATED REC PLOT
3 REM IN MULTI-RES
4 REM-----
5 HIRES 0,2:MULTI 2,4,6
10 X=1:Y=1:A=90:B=180
20 REC X,Y,A,B,4

```

```

30 X=X+3:Y=Y+1:A=A-1:B=B-2
35 IF X>100 THEN 10
40 GOTO 20

```

---

```

1 REM PROGRAM 65
2 REM ANIMATED LINE PLOT
3 REM IN MULTI-RES
4 REM-----
5 HIRES 0,1:MULTI 4,6,2
10 X=200:Y=0:Z=1
20 LINE 0,0,X,Y,Z
30 X=X+2:Y=Y+3:Z=Z+1
40 IF Z=4 THEN Z=1
50 IF Y>250 THEN 5
60 GOTO 20

```

---

```

1 REM PROGRAM 66
2 REM MORE MULTICOLORS
3 REM IN MULTI-RES
4 REM-----
10 HIRES 0,7:MULTI 2,4,6
20 REC 5,5,90,90,2
30 REC 10,10,30,30,1
40 REC 25,25,40,40,3
50 GOTO 50

```

---

In multi-res, each pixel is twice as wide as it appears in hi-res. As a result, multi-res has half the horizontal resolution of hi-res. Still, multi-res has a pretty respectable look, and the tradeoff results in the ability to put multicolors on the screen.

## Plot Types in Multi-Res

Plot types work slightly differently in multi-res than they do in hi-res, to account for the additional colors available. A plot type of 0 still functions to clear a dot. A plot type of 1 plots a dot in color 1. A plot type of 2 plots in color 2, and a plot type of 3 plots in color 3.

If you specify a plot type of 4, the plot will inverse the dot color in the following fashion:

```

color 0 changes to color 3
color 1 changes to color 2
color 2 changes to color 1
color 3 changes to color 0

```



This plot type can give us animated rainbow effects for a small expenditure of code.

Creative use of plot types can make animation a cinch. Here are some starting points:

---

```

1 REM PROGRAM 67
2 REM ANIMATION USING PLOT TYPE
3 REM IN MULTI-RES
4 REM-----
10 HIRES 0,1:MULTI 2,4,7
20 X=10:Y=20
30 REC X,Y,X+10,Y+10,K
40 FOR Z=1 TO 10:NEXT Z
50 X=X+2:Y=Y+3:K=K+1
60 IF K=4 THEN K=1
70 IF Y>90 THEN 10
80 GOTO 30

```

---



---

```

1 REM PROGRAM 68
2 REM MORE ANIMATION USING PLOT TYPE
3 REM IN THE MULTI-RES MODE
4 REM-----
5 HIRES 0,1:MULTI 0,7,5
10 X=100:Y=0
20 LINE 0,0,X,Y,Z
30 X=X+2:Y=Y+3:Z=Z+1
40 IF Z=4 THEN Z=1
50 IF X>250 THEN 10
60 GOTO 20

```

---



---

```

1 REM PROGRAM 69
2 REM STILL MORE ANIMATION
3 REM IN THE MULTI-RES MODE
4 REM-----
5 HIRES 1,6:MULTI 4,1,6
10 X=10
20 REC X,X+10,X+15,X+20,1
30 FOR Z=1 TO 10:NEXT Z
40 REC X,X+10,X+15,X+20,0
50 FOR Z=1 TO 10:NEXT Z
60 REC X,X+10,X+15,X+20,2
70 FOR Z=1 TO 10:NEXT Z
80 REC X,X+10,X+15,X+20,0
90 X=X+1:IF X=70 THEN 10
100 GOTO 20

```

---

## LOW COL

LOW COL enables you to specify a new and different set of graphics plotting colors from those originally selected with the HIRES or MULTI commands. The format for the command is

LOW COL color 1,color 2,color 3

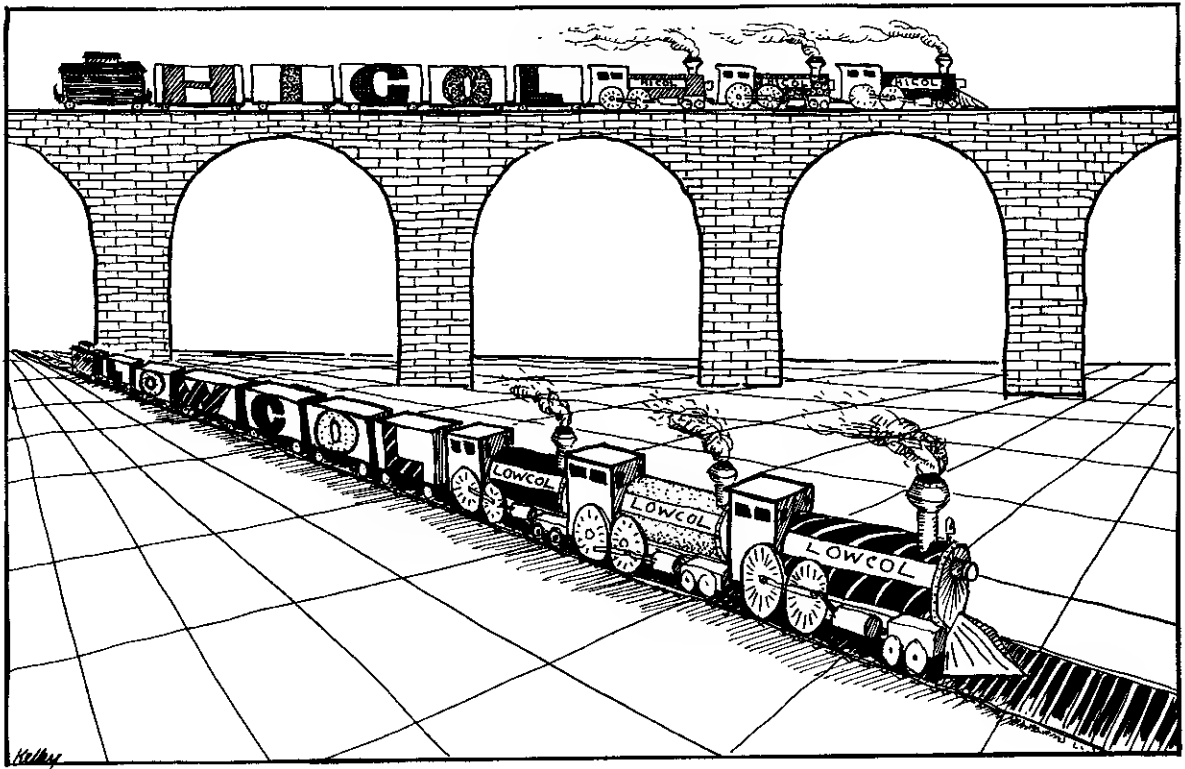
where:

color 1 = 0 – 15

color 2 = 0 – 15

color 3 = 0 – 15

Note please that because only two colors are used in hi-res graphics plotting, the third color in the LOW COL command will have no effect. However, you must still specify three parameters, even when using LOW COL for a hi-res application. Specify a third “dummy” parameter for hi-res, even though it will be inoperative. When working in multi-res, choose all three colors available.



## HI COL

HI COL allows you to revert to the originally selected plotting colors. With it you can restore the original colors set up with a HIRES or MULTI command, after a LOW COL command has been invoked.

Here are examples of LOW COL and HI COL in hi-res and multi-res applications:

---

```

1 REM PROGRAM 70
2 REM LOW COL AND HI COL
3 REM
4 REM-----
10 HIRES 0,7:MULTI 0,1,2
30 REC 20,20,30,30,1
40 REC 25,25,35,35,2
50 REC 30,30,40,40,3
60 LOW COL 3,4,5
70 REC 35,35,45,45,1
80 REC 40,40,50,50,2
90 REC 45,45,55,55,3
99 GOTO 99

```

---



---

```

1 REM PROGRAM 71
2 REM LOW COL AND HI COL
3 REM ANOTHER EXAMPLE
4 REM-----
5 X=100:Y=0
10 HIRES 0,1:MULTI 2,6,7
20 LINE 0,0,X,Y,Z
30 X=X+2:Y=Y+2:Z=Z+1
40 IF Z=4 THEN Z=1
50 IF Y>80 THEN LOW COL 0,1,5
60 IF Y>120 THEN HI COL
70 IF Y>200 THEN 5
80 GOTO 20

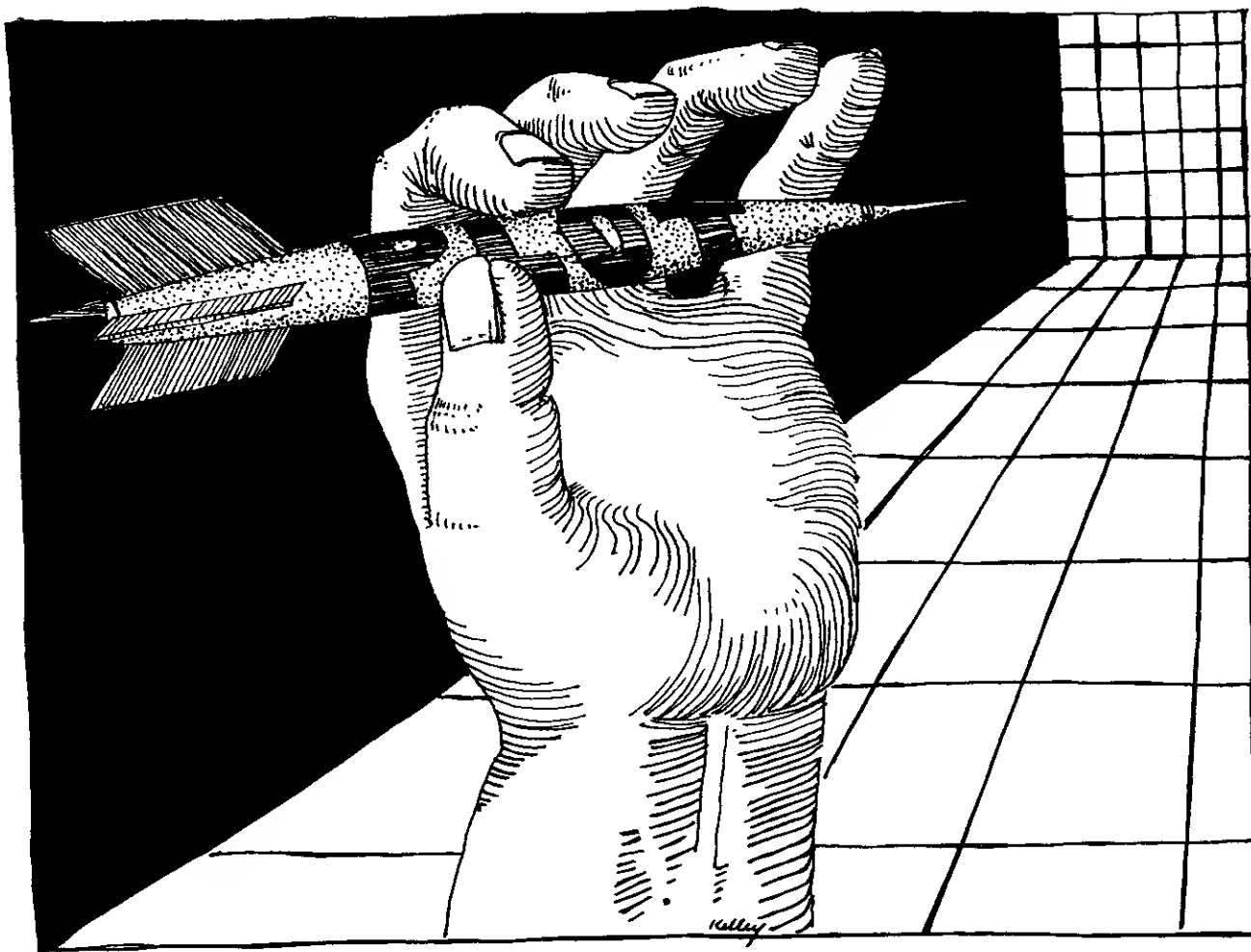
```

---

## PLOT

PLOT very simply allows you to plot a dot. The format for the command is as follows

PLOT x,y,plot type



You can use the PLOT command to draw a starfield or to plot a shape. Here are examples of both.

---

```
1 REM PROGRAM 72
2 REM THE PLOT COMMAND
3 REM
4 REM-----
10 HIRES 1,0
15 PLOT 51,28,1
20 PLOT 96,40,1
30 PLOT 140,72,1
40 PLOT 176,101,1
50 PLOT 226,129,1
60 PLOT 193,154,1
70 PLOT 153,128,1
99 GOTO 99
```

---

---

```

1 REM PROGRAM 73
2 REM THE PLOT COMMAND
3 REM ANOTHER EXAMPLE
4 REM-----
10 HIRES 0,7:MULTI 2,4,6
20 X=10:Y=10
30 PLOT X,Y,1
40 X=X+2:Y=Y+1
50 IF X<100 THEN 30
60 PLOT X,Y,1
70 Y=Y+2
80 IF Y<140 THEN 60
90 PLOT X,Y,1
100 X=X-2
110 IF X>10 THEN 90
120 PLOT X,Y,1
130 Y=Y-2
140 IF Y>10 THEN 120
150 GOTO 150

```

---

## TEST

The command TEST allows you to determine if a plot has been drawn at a screen location where another plot exists. Using the command, you can examine the status of a location on a graphics screen. The format is

variable= TEST (x,y)

where:

variable = legal variable named by user

x = legal x coordinate

y = legal y coordinate

The parameters x and y are the screen coordinates of the point being tested. If a dot has been plotted at that point, the plot type of the dot is returned. A value of 0 is returned if no dot is present. The dot may be any part of a graphics shape, plotted in hi-res or multi-res.

The real value of the TEST command is to detect where one plot intersects another. Here is an example of the command at work:

---

```

1 REM PROGRAM 74
2 REM THE TEST COMMAND
3 REM TO DETECT AN INTERSECTION
4 REM-----

```

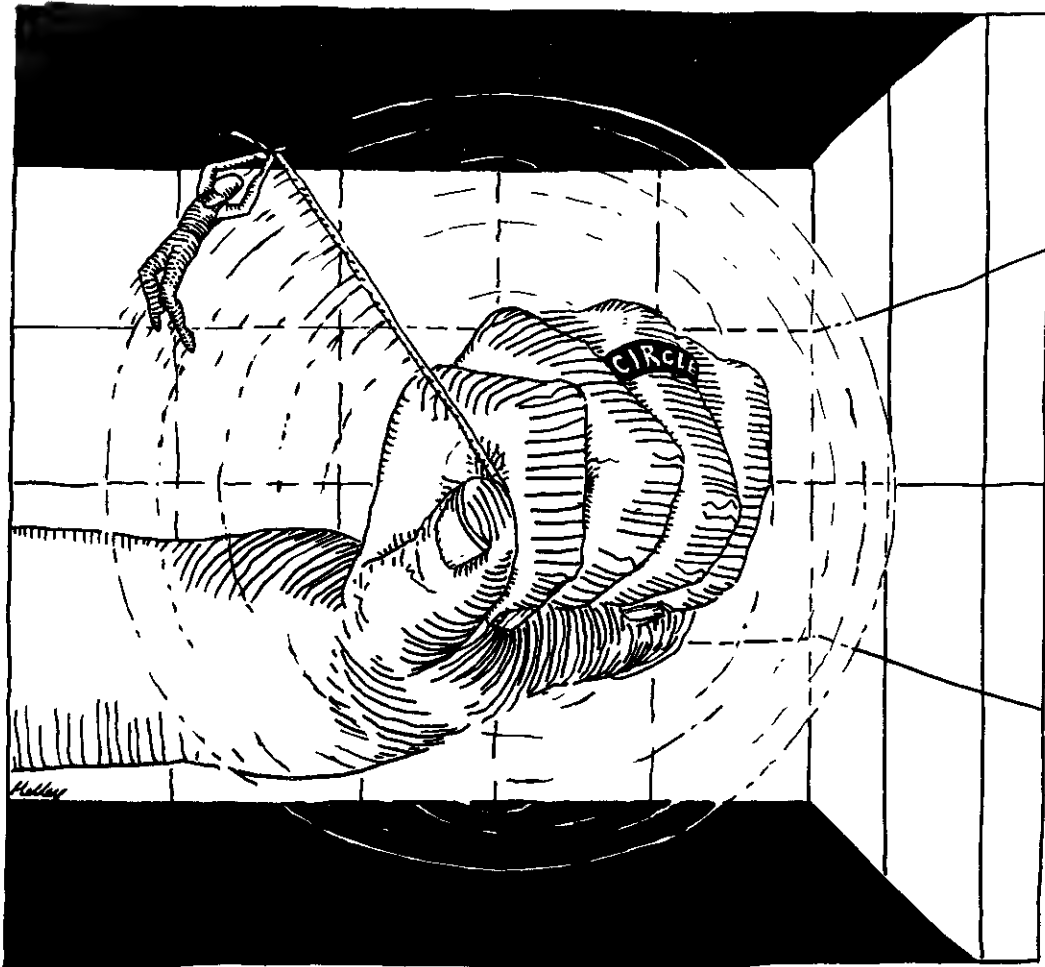
---

```
10 HIRES 0,1:MULTI 2,4,6
20 FOR X=20 TO 140
30 PLOT X,20,1:PLOT X,60,1
40 NEXT:X=20:Y=60
50 IF TEST(X,Y)=0 THEN PLOT X,Y,2
60 X=X+1:Y=Y-1
70 IF TEST(X,Y)=0 THEN 50
80 IF TEST(X,Y)=0 THEN PLOT X,Y,2
90 X=X+1:Y=Y+1
100 IF TEST(X,Y)=0 THEN 80
110 IF TEST(X,Y)=0 THEN PLOT X,Y,2
120 X=X+1:Y=Y-1
130 IF TEST(X,Y)=0 THEN 110
140 GOTO 140
```

RENDERING

**CIRCLE**

You've got it. CIRCLE draws a circular shape. The format for the command is



CIRCLE cent x,cent y,x radius,y radius,plot type

The parameters cent x and cent y specify the center of the circle. The parameters x radius and y radius indicate the horizontal and vertical radii of the shape. By varying these radii, circles and ellipses of different sizes can be drawn.

When you first begin plotting circles, you may find yourself somewhat perplexed. Because screen pixels are somewhat wider than they are tall, equal horizontal and vertical radii (x radius = y radius) will not result in a perfect circle on the screen. In order to get a good circle in hi-res mode, the x radius must be 1.4 times the y radius in length. In multi-res, the x radius must be 1.6 times the y radius to achieve a good-looking circle. Circular shapes that have equal horizontal and vertical radii will plot as ellipses.

Here are some circles and ellipses to get you started.

---

```

1 REM PROGRAM 75
2 REM THE CIRCLE COMMAND
3 REM
4 REM-----
10 HIRES 0,7
20 CIRCLE 150,80,70,50,1
30 GOTO 30

```

---



---

```

1 REM PROGRAM 76
2 REM THE CIRCLE COMMAND
3 REM USED TO DRAW AN ELLIPSE
4 REM-----
10 HIRES 0,7
20 CIRCLE 150,80,90,50,1
30 GOTO 30

```

---



---

```

1 REM PROGRAM 77
2 REM THE CIRCLE COMMAND
3 REM AND COLOR ANIMATION EFFECTS
4 REM-----
5 HIRES 0,7:MULTI 2,5,6
10 X=60:Y=80:Z=1
20 CIRCLE 80,80,X,Y,Z
30 X=X-3:Y=Y-3:Z=Z+1
40 IF Z=4 THEN Z=1
50 IF X>10 THEN 20
60 GOTO 60

```

---

---

```

1 REM PROGRAM 78
2 REM THE CIRCLE COMMAND
3 REM USED FOR ANIMATED MOIRE DESIGN
4 REM-----
5 HIRES 0,1
10 X=100:Y=76
20 CIRCLE 130,80,X,Y,1
30 CIRCLE 150,100,X,Y,1
40 X=X-3:Y=Y-3
50 IF Y>10 THEN 20
60 GOTO 60

```

---

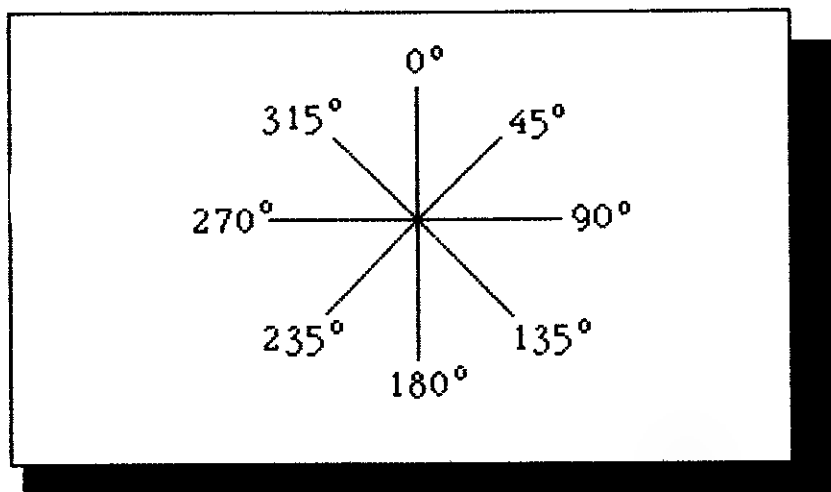
## ARC

The ARC command allows you to draw part of the circumference of a circular shape, without having to draw the entire circle and then trying to erase. Format for the command is as follows:

ARC cent x,cent y,beg ang ,end ang,interval,x radius,y radius,plot type

The parameters cent x and cent y are the screen coordinates of the center of the circular shape from which the arc is drawn. Parameters beg ang and end ang define the start and end angles of the arc. To determine angles, use the following chart:

Figuring the angles.





The parameter interval specifies the plotting increment, in degrees between each point on the arc. This determines whether the arc will be plotted as a solid or jagged line. To obtain an arc drawn with a solid line, this value should be set to 1. A larger value results in jagged lines of increasing distance between dots.

Parameters x radius and y radius indicate the horizontal radius and vertical radius of the circular shape of which the arc would be a part if the entire circle were to be drawn.

Try these to get a feel for the ARC command:

---

```

1 REM PROGRAM 79
2 REM THE ARC COMMAND
3 REM
4 REM-----
10 HIRES 0,7
20 ARC 150,80,0,235,1,60,40,1
30 GOTO 30

```

---



---

```

1 REM PROGRAM 80
2 REM THE "ARC" COMMAND
3 REM ANOTHER EXAMPLE
4 REM-----
10 HIRES 0,7
20 ARC 150,80,0,270,40,60,40,1
30 GOTO 30

```

---

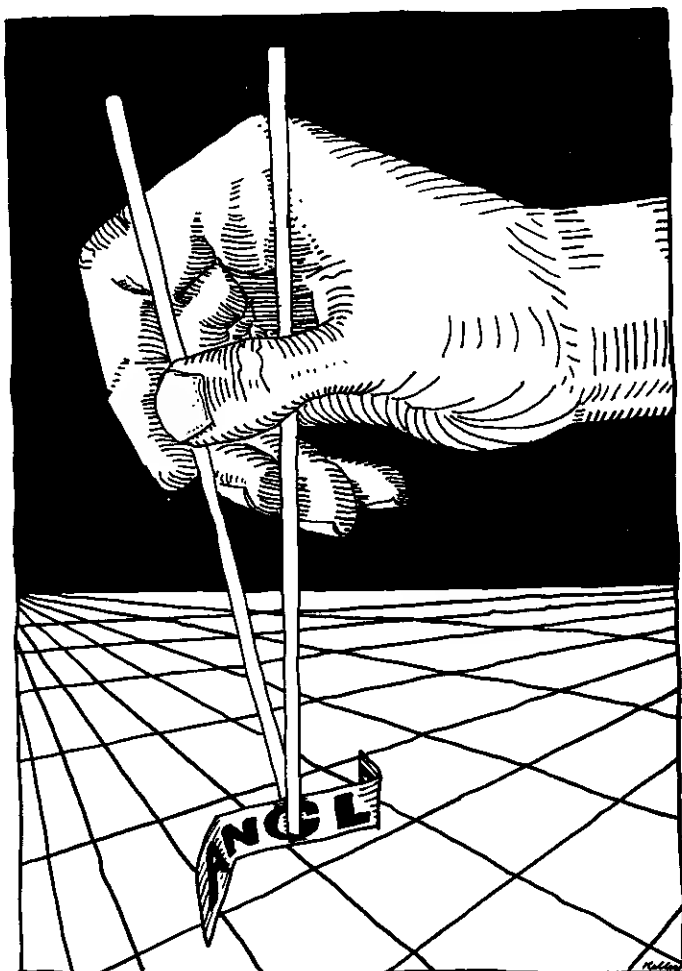
## ANGLE

The command ANGLE allows you to draw the radius of a circle, without having to display its circumference. This may seem exactly the same as drawing a straight line, and at first glance it is. But the ANGLE command allows you to plot multiple radii from a common center. This makes the design of star shapes, pie charts, and spoked wheels quite simple.

The format of the command is

ANGL cent x,cent y,angle,x radius,y radius,plot type

The parameters cent x and cent y are, as usual, the screen coordinates of the center of the circle. Angle is the angle, in degrees, at which the radius



is depicted. Again, you may use the chart provided above to figure out angle coordinates. Parameters x radius and y radius are again the horizontal and vertical radii of the circle they would define, if we were to plot it.

Here are some examples.

---

```

1 REM PROGRAM 81
2 REM THE ANGL COMMAND
3 REM
4 REM-----
10 HIRES 0,7
20 ANGL 150,80,250,60,60,1
30 ANGL 150,80,90,60,60,1
40 GOTO 40

```

---

```

1 REM PROGRAM 82
2 REM THE ANGL COMMAND

```

---

```

3 REM ANOTHER EXAMPLE
4 REM-----
10 HIRES 0,7
20 X=0
30 ANGL 150,80,X,60,60,1
40 X=X+15
50 IF X<360 THEN 30
60 GOTO 60

```

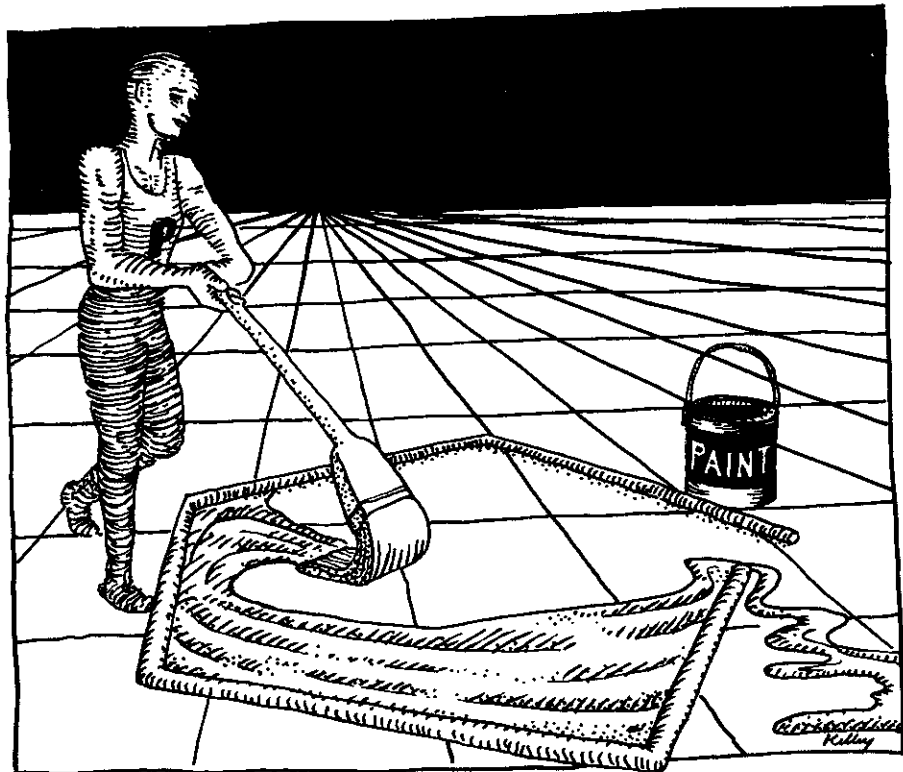
```

1 REM PROGRAM 83
2 REM THE ANGL COMMAND
3 REM A MULTI-RES EXAMPLE
4 REM-----
10 HIRES 0,7:MULTI 2,6,7
20 X=0:Z=1
30 ANGL 80,80,X,60,60,Z
40 X=X+3:Z=Z+1
50 IF Z=4 THEN Z=1
60 IF X<360 THEN 30
70 GOTO 70

```

## PAINT

PAINT is a very powerful command, and one you will be using often from the hi-res and multi-res modes. Paint fills an enclosed area with color. Using the



PAINT command, you can get the stick-figure plots available from the graphics mode to fill right out into solid shapes.

Please note one extremely important fact: the area to be colored in must be *completely* enclosed or the fill will "spill out" into the background color. The area to be painted is specified by the x and y coordinates of any point within its closed boundaries.

The format for using the command is

PAINT x,y,plot type

In the hi-res mode, the same area may be painted only once. Trying to paint over it will not result in any action. In multi-res, however, the same area may be repainted as many times as you like.

The plot type in this command must be 0,1,2,or 3 only. The color of the fill is determined by the plot type. You can't fill a multi-res shape with inverse color.

Here are some examples of PAINT:

```
1 REM PROGRAM 84
2 REM THE PAINT COMMAND
3 REM
4 REM-----
10 HIRES 0,5:MULTI 2,6,4
20 REC 10,5,35,60,1
30 CIRCLE 90,50,35,40,2
40 REC 20,60,70,90,3
50 PAINT 11,6,1
60 PAINT 90,50,2
70 PAINT 21,66,3
99 GOTO 99
```

---

```
1 REM PROGRAM 85
2 REM THE PAINT COMMAND
3 REM ANOTHER EXAMPLE
4 REM-----
10 HIRES 0,5:MULTI 5,6,7
20 CIRCLE 90,50,50,40,1
30 CIRCLE 80,60,50,40,2
40 CIRCLE 70,70,50,40,3
50 CIRCLE 60,80,50,40,1
60 REM PAINT 89,49,1
70 PAINT 79,39,2
80 PAINT 69,29,3
90 PAINT 59,20,1
100 PAINT 89,90,1
```

```

110 PAINT 99,100,2
120 PAINT 79,110,3
130 GOTO 130

```

```

1 REM PROGRAM 86
2 REM PAINTING OVER THE SAME AREA
3 REM HIT RUN/STOP-RESTORE TO STOP
4 REM-----
10 HIRES 0,1:MULTI 7,8,9
20 REC 30,30,50,70,1
30 X=1
40 PAINT 31,31,X
50 X=X+1:IF X=4 THEN X=1
60 GOTO 40

```

## BLOCK

BLOCK works very much like PAINT, but it allows you in one step to draw a fully-shaded block of color. Using the BLOCK command is very similar to using a REC command followed by a PAINT of the same color inside it. BLOCK draws a rectangle and fills it with color at the same time. In the BLOCK command the color of the rectangle and the fill color are always exactly the same.

The format for the command is

BLOCK beg x,beg y, end x,end y,plot type

The BLOCK command is useful if you wish to create several adjacent blocks of color without separating them by lines. The parameters beg x and beg y specify the top left corner of the block, while end x and end y specify the lower right corner.

Here is BLOCK at work:

```

1 REM PROGRAM 87
2 REM THE BLOCK COMMAND
3 REM
4 REM-----
10 HIRES 2,3:MULTI 3,4,5
20 X=10:Y=60:Z=1
30 BLOCK X,X,Y,Y,Z
40 X=X+10:Y=Y+10:Z=Z+1
50 IF Z=4 THEN Z=1
60 IF Y<200 THEN 30
70 GOTO 70

```

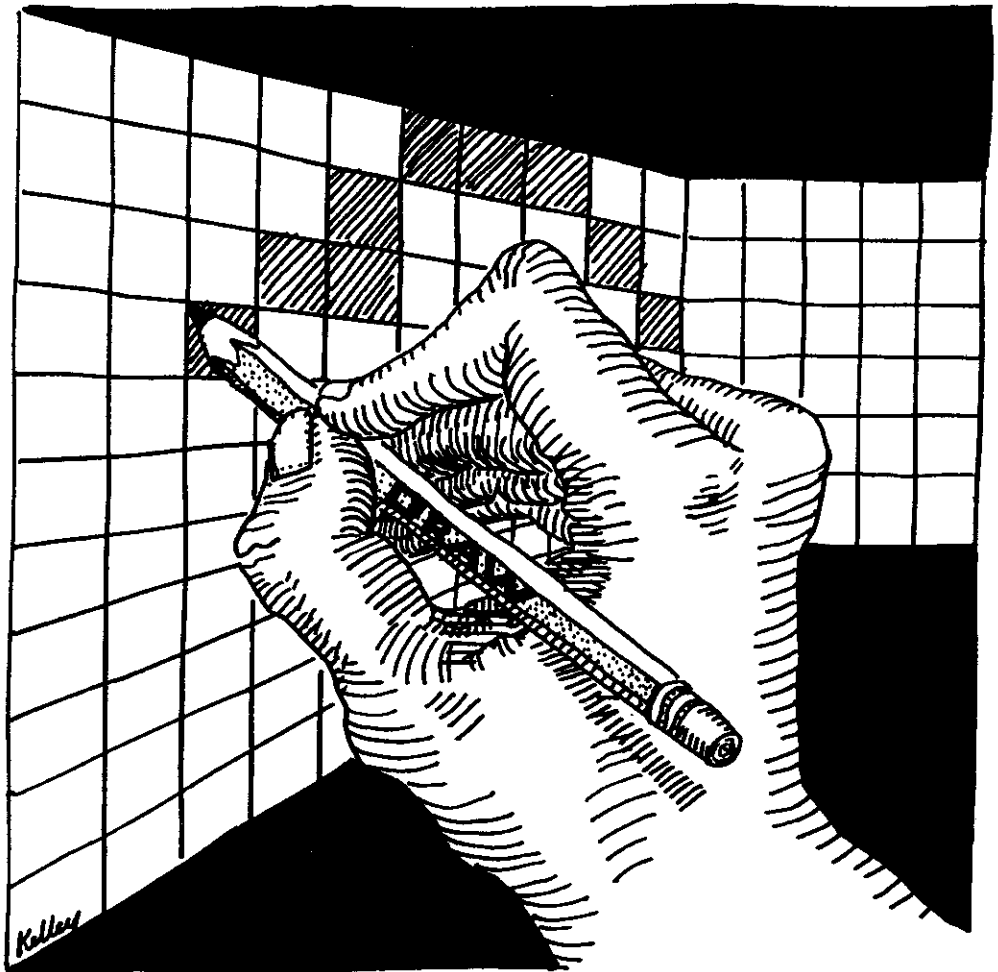
---

```
1 REM PROGRAM 88
2 REM THE BLOCK COMMAND
3 REM ANOTHER EXAMPLE
4 REM-----
10 HIRES 2,3:MULTI 4,5,6
20 X=5:Y=150:Z=1
30 BLOCK X,Y,Y,X,Z
40 X=X+2:Y=Y-2:Z=Z+1
50 IF Z=4 THEN Z=1
60 IF Y>3 THEN 30
70 GOTO 70
```

---

## **DRAW**

The DRAW command allows you to design a shape, save it as a "string," and then display it on the screen in any of several ways. The shape is designed in the same way as drawing a picture on a piece of paper—and in fact resembles some aspects of LOGO turtle graphics quite closely.



There are nine commands available to the pen in a draw command. They are the following:

- 0—move 1 pixel to the right
- 1—move 1 pixel up
- 2—move 1 pixel down
- 3—move 1 pixel to the left
- 5—move 1 pixel to the right and plot a dot
- 6—move 1 pixel up and plot a dot
- 7—move 1 pixel down and plot a dot
- 8—move 1 pixel to the left and plot a dot
- 9—stop drawing

The format for the DRAW command is

DRAW "pen commands",x,y,plot type

Each number within quotes is a DRAW command number which tells the computer what to draw and how to draw it. A maximum of 74 instructions can be placed within the quotation marks on any one program line. You may, however, add strings of instructions together up to a maximum of 255. To continue the shape in a following command, a new origin must be specified beginning where the old one ended off.

Using the DRAW command is a bit tricky at first and rather cumbersome even after you have mastered the technique. Still, you can do some neat things with shapes created using the DRAW command, as we shall see.

---

```

1 REM PROGRAM 89
2 REM THE "DRAW" COMMAND
3 REM YOU *MUST* USE ROT COMMAND TOO
4 REM-----
10 HIRES 0,1:ROT 0,2
20 DRAW "55555555557777777777888888888866
666666666",160,50,1
30 GOTO 30

```

---



---

```

1 REM PROGRAM 90
2 REM ANIMATION WITH THE "DRAW" COMMAND
3 REM ROT AND DRAW ALWAYS WORK TOGETHER
4 REM-----
10 HIRES 1,0:ROT 2,1:X=20
20 A$="5555555555777777777788888888886666
666666666"
30 DRAW A$,X,20,1

```

---

```

50 DRAW A$,X,20,0
60 X=X+1:IF X=250 THEN X=20
70 GOTO 30

```

---

```

1 REM PROGRAM 91
2 REM ABSTRACT ANIMATION WITH "DRAW"
3 REM
4 REM-----
10 HIRES 1,0:MULTI 2,4,6:ROT 1,4
20 A$="5555555555555555755555577777777777
77666666666666666666665656565"
30 X=0:Y=0
40 DRAW A$,X,Y,1
50 X=X+2:Y=Y+1:IF Y<100 THEN 40
55 X=0:Y=10:ROT 2,4
60 DRAW A$,X,Y,2
70 X=X+2:Y=Y+1:IF Y<100 THEN 60
80 X=0:Y=5:ROT 3,4
90 DRAW A$,X,Y,3
100 X=X+1:Y=Y+2:IF Y<255 THEN 90
110 GOTO 110

```

---

## **ROT**

The command ROT is short for "rotation" and allows you to display a shape created by the DRAW command at a specified angle of rotation and in a specified size. The command format is

ROT rot angle,size

The parameter rot angle specifies by how much the shape is to be rotated and it uses the same angle notation as ARC and ANGLE. This value of rot angle (range is limited to 0 through 7) defines the angle of rotation as shown in the table below:

Rotation	0— 0 degrees
	1— 45
	2— 90
	3— 135
	4— 180
	5— 225
	6— 270
	7— 315



The second parameter in the ROT command defines the size of the shape you have designed. A "1" in this position indicates that the shape is to be displayed at normal size: where each number in the DRAW command represents a movement of one pixel. Increasing this number results in an increased figure size. A "2" in this position will move 2 pixels per DRAW command, and so on.

Remember that a design will disappear from the screen if it is specified to be too large. Keep the size of the DRAW figure within acceptable screen limits.

---

```

1 REM PROGRAM 92
2 REM SPINNING ROT VALUES
3 REM
4 REM-----
10 HIRES 1,0:MULTI 2,3,4:X=0
20 A$="6666666666666666665555555777779"
30 ROT X,1:DRAW A$,70,99,1:FOR Z=1 TO 50
:NEXT Z
35 DRAW A$,70,99,0
40 X=X+1:IF X=8 THEN X=0
50 GOTO 30

```

---



---

```

1 REM PROGRAM 93
2 REM ROT VALUES--ANOTHER EXAMPLE
3 REM
4 REM-----
10 HIRES 2,1
20 A$="5555555555757575757575"
30 ROT 0,1:DRAW A$,20,20,1
40 ROT 1,2:DRAW A$,40,30,1
50 ROT 2,3:DRAW A$,90,50,1
60 ROT 3,4:DRAW A$,160,70,1
70 ROT 4,5:DRAW A$,190,150,1
80 GOTO 80

```

---



---

```

1 REM PROGRAM 94
2 REM ROT VALUES--AN ANIMATED EXAMPLE
3 REM
4 REM-----
10 HIRES 2,1:MULTI 0,3,4:X=20
20 A$="555577775555777788887777666677776
666888866655556666"
30 ROT R,1:DRAW A$,X,50,1

```

---

---

```

32 FOR Z=1 TO 50:NEXT Z
35 ROT R,1:DRAW A$,X,50,0
40 X=X+5:R=R+1:IF R=8 THEN R=0
50 IF X>150 THEN 10
60 GOTO 30

```

---

## DISPLAYING TEXT IN HI-RES AND MULTI-RES

When you are composing hi-res and multi-res screens, it is sometimes desirable to print text alongside pictures. Simon's Basic has two commands which allow you to do just that.

### CHAR

The CHAR command allows you to display a text character on a hi-res or multicolor graphics screen. The format for the command is

CHAR x,y,character code,plot type,size

The parameters x and y specify the location of the character on the screen. The next parameter in the command is the character code of the character you wish to display (a list of character poke codes appears in the Commodore Programmer's Reference Guide).

The last parameter in this command specifies the height of the character. A parameter of 1 will display text at its normal graphics size, which is eight pixels high. Increasing the parameter acts just like the size parameter in a ROT command, but for one difference: the characters expand only vertically—their width cannot be varied.

---

```

1 REM PROGRAM 95
2 REM THE CHAR COMMAND
3 REM
4 REM-----
10 HIRES 0,7
20 CHAR 30,40,1,1,1
30 GOTO 30

```

---



---

```

1 REM PROGRAM 96
2 REM THE CHAR COMMAND
3 REM ANOTHER EXAMPLE
4 REM-----
10 HIRES 2,1:MULTI 2,3,4
15 REC 75,60,80,90,2
20 CHAR 77,70,11,1,2
30 GOTO 30

```

---

---

```

1 REM PROGRAM 97
2 REM THE CHAR COMMAND
3 REM ONE MORE EXAMPLE
4 REM-----
10 HIRES 2,1:MULTI 2,3,4
20 CHAR 20,40,1,1,5
30 CHAR 30,50,2,2,5
40 CHAR 40,60,3,3,5
50 GOTO 50

```

---

## TEXT

The TEXT command allows you to print character strings on graphics screens. The format for use of the command is

TEXT x,y," CTRL-A or CTRL-B character string",plot type,size,spacing

The parameters x and y specify the screen coordinates of the first letter of the string. The next parameter is the string itself.

The control character preceding the string, either CTRL-A or CTRL-B, indicates whether the string is to be displayed in upper or lower case letters. You can also mix upper and lower case letters in a single string expression. Precede all upper case letters with CTRL-A, and all lower case letters with CTRL-B. *on a 640x480 screen, the string "HEY THERE!" will be 100 pixels wide.*

The size parameter acts exactly as it does in the CHAR command. The parameter spacing determines how much pixel space will be inserted between characters in a string. Default for this value is 8. Increasing this parameter increases the space between characters.

---

```

1 REM PROGRAM 98
2 REM THE "TEXT" COMMAND
3 REM
4 REM-----
10 HIRES 2,1:MULTI 2,3,4
20 TEXT 20,20,"HEY THERE!",1,2,7
30 GOTO 30

```

---



---

```

1 REM PROGRAM 99
2 REM THE "TEXT" COMMAND
3 REM ANOTHER EXAMPLE
4 REM-----
10 HIRES 2,1:MULTI 2,6,7
20 TEXT 5,60,"HOW ARE THINGS GOING?",2,1,7
30 GOTO 30

```

---

---

```

1 REM PROGRAM 100
2 REM THE "TEXT" COMMAND
3 REM IN CONCERT WITH OTHER SHAPES
4 REM-----
10 HIRES 2,1:MULTI 2,6,7
20 BLOCK 50,80,140,160,2
30 CIRCLE 50,70,50,70,1
40 PAINT 21,21,1
50 PAINT 51,81,3
60 TEXT 20,30,"CIRCLE",3,1,7
70 TEXT 70,150,"RECTANGLE",3,1,7
99 GOTO 99

```

---

## NRM

The command NRM stands for "normal" or "normal resolution mode," and allows you to return to a low-res or text screen from a graphics screen. Since both screens reside separately in screen memory, the NRM command will return to the low-res screen that existed before a HIRES command was invoked.

## CSET

The command CSET allows you to go in the other direction, recalling the last graphics screen from the text mode. The format for the command is

CSET 2

---

```

1 REM PROGRAM 101
2 REM THE "NRM" COMMAND
3 REM
4 REM-----
10 HIRES 2,1:MULTI 9,12,15
20 BLOCK 50,80,140,160,2
30 CIRCLE 50,70,50,70,1
60 TEXT 20,30,"CIRCLE",2,1,7
70 TEXT 70,150,"RECTANGLE",1,1,7
80 PAUSE 3:NRM
90 PRINT"NOW BACK TO THE TEXT PAGE."

```

---



---

```

1 REM PROGRAM 102
2 REM THE "CSET" COMMAND
3 REM TO RETURN FROM A "NRM"
4 REM-----

```

---

```
10 HIRES 2,1:MULTI 9,12,15
20 BLOCK 50,80,140,160,2
30 CIRCLE 50,70,50,70,1
60 TEXT 20,30,"CIRCLE",2,1,7
70 TEXT 70,150,"RECTANGLE",1,1,7
80 PAUSE 3:NRM
90 PRINT"WE INTERRUPT THIS PROGRAM..."
100 PAUSE 3
110 CSET2:MULTI 2,5,7
120 GOTO 120
```

.....



## MOBBING UP IN SIMON'S BASIC

It would be tough for you to own a C-64 very long without knowing that your machine can generate "sprites." A sprite is a piece of color screen data that can be controlled independently from the "normal" screen display. The screen data can be manipulated to move, change color, and even change shape without the need to recalculate backgrounds. Up to eight simultaneous and independent sprite shapes can be programmed in a relatively straightforward manner.

### SPRITES ARE MOBS

First of all, let's get one piece of terminology straight. In Simon's Basic, a sprite is not a sprite—it is a MOB: a movable object block. So get the word "sprite" out of your head—it's MOB from here on in.

MOBs don't care whether you display them on hi-res or low-res screens (although you need to do a bit of extra work to get them to appear on hi-res and multi-res screens). You can display a MOB in any single color in the hi-res mode, and in up to three different colors in the multicolor mode. A hi-res MOB is 24 pixels wide by 21 pixels deep. Remember that in multicolor mode pixels are double width, so multicolor MOBs are 12 dots wide by 21 dots deep.

Before your enthusiasm takes you off the deep end, you should realize that working with MOBs can be tricky. There are many things to keep track of and dozens of ways to trip up.

On the bright side, it is much easier to deal with them from Simon's Basic than it is from plain old Basic. Simon's Basic has many special commands specifically designed to make working with MOBs as simple as possible. There is no comparison between learning to handle MOBs from Simon's Basic and controlling "sprites" the hard way.

## DESIGN

The purpose of the DESIGN command is to allocate memory space for a MOB. Before you do much else, you must tell the computer what kind of MOBs you are designing, and where they shall reside in memory. The format for the DESIGN command is as follows:

DESIGN mob-res,address

where:

mob-res = 0 or 1

address = 2048 — 4095 (in multiples of 64)





The first parameter tells whether you are designing a hi-res or multi-res MOB. If you specify 0 for this parameter, you are designing a hi-res MOB. If you specify 1, you are designing a multi-res MOB. The second parameter tells the computer where to look in memory to find this specific MOB shape.

Because each MOB uses up 64 bytes of memory, each block of available MOB memory is 64 bytes long. MOB memory starts at memory location 2048, and proceeds from there in blocks of 64. The table below shows a MOB memory map.

MOB memory map.

MOB	Block	Memory Location	
0	32	2048	=2048
1	33	2048+64	=2112
2	34	2048+128	=2176
3	35	2048+192	=2240
4	36	2048+256	=2304
5	37	2048+256+64	=2368
6	38	2048+256+128	=2432
7	39	2048+256+192	=2496
8	40	2048+256+256	=2560

Things start to get a little sticky now, but try to hang on. There is a block number associated with each block of MOB storage space. In the table above, we can see that the block numbers start at 32, which is associated with the block that begins at memory location 2048. Block 33 starts at location  $2048+64$ , block 34 at  $2048+64+64$ , and so on. To determine what block number is associated with any given memory location, simply divide the memory location by 64.

Let's answer a quick question probably now occurring in many minds—if only 8 MOBS can be displayed at a time, why designate so much room for MOB addresses?

Well even though we are limited to displaying eight or fewer MOBs at a time, we may want to define many more MOBs than that. Once we do, we can change them on the fly—and this is how we animate MOBs. By switching slightly different MOBs quickly, we can make a shape appear to move: we can make a runner run and a jumper jump. Then when we move the MOBs while flipping between them, we have a bona fide moving picture. That's just one

reason why we sometimes keep lots more than eight MOB's defined in memory.

Let's go back to the hard-to-digest MOB rules, which will only become clear through experience with them.

If a MOB is to be used on a hi-res graphics screen, you must add a graphics constant value of 49152 to the screen address figure. When calculating a block number, do not take the 49152 graphics constant into consideration. Merely work the calculation as usual—49152 is a flag that allows the MOB to appear on a hi-res or multi-res screen. If you forget this when putting a sprite on a hi-res or multi-res screen, you'll go nuts trying to figure out why it's not working.

@

The @ command is the header that indicates a MOB design grid is to follow. You will use a design grid to construct MOB shapes. The grid is 24 dots wide when you are designing a hi-res mob, and 12 dots wide when you are designing a multicolor mob.

Something else important: you should ensure that each line number for the grid is the same length—for example, three digits or four digits. By doing this, you will keep the indentation of points on the grid constant, and not encounter problems during the MOB design process.

It is a good idea to keep design grid template files on disk for hi-res and multi-res MOB's. Then, when it comes time to design a MOB, you can load the table and proceed with your design.

Here are hi-res and multi-res MOB design grid templates:

---

```

1 REM PROGRAM 103
2 REM HIRES MOB TEMPLATE
3 REM
4 REM-----
100 @.....
110 @.....
120 @.....
130 @.....
140 @.....
150 @.....
160 @.....
170 @.....
180 @.....
190 @.....
200 @.....
210 @.....
220 @.....
230 @.....
240 @.....
250 @.....
260 @.....
270 @.....
280 @.....

```

---

```

290 @.....
300 @.....

```

---



---

```

1 REM PROGRAM 104
2 REM MULTI-RES MOB TEMPLATE
3 REM
4 REM-----
100 @.....
110 @.....
120 @.....
130 @.....
140 @.....
150 @.....
160 @.....
170 @.....
180 @.....
190 @.....
200 @.....
210 @.....
220 @.....
230 @.....
240 @.....
250 @.....
260 @.....
270 @.....
280 @.....
290 @.....
300 @.....

```

---

Creating multiple MOBs? No problem. Use the screen editor to change the line numbers on blank design grids. In this way you can clone as many new grids as you need in any simple program—using one grid template from disk.

When you are designing a monochrome MOB in the hi-res mode, the color code character to use when “filling in the blanks” is B. That will be the color assigned in the MOB SET command, which we’ll learn about just up ahead. So the @ grid for a hi-res MOB might take the following form:

---

```

1 REM PROGRAM 105
2 REM HIRES MOB EXAMPLE
3 REM
4 REM-----
100 @.....
110 @.....
120 @.....

```

---

```

130 @.....
140 @.....BBBBBB.....
150 @.....BBBBBBBBBBBBBB.....
160 @...BB.BBBBBBBBBBBB.BB....
170 @..BBBB.BBBBBBBBBB.BBBB...
180 @.BBBBBB.BBBBBB.BBBBBB...
190 @.BBBBBBB.BBBB.BBBBBBB...
200 @.BBBBBBB.BBBB.BBBBBBB...
210 @.BBBBBBB.BBBB.BBBBBBB...
220 @.BBBBBBB.BBBBBB.BBBBBB...
230 @..BBBB.BBBBBBBBBB.BBBB...
240 @...BB.BBBBBBBBBBBB.BB....
250 @.....BBBBBBBBBBBBBB.....
260 @.....BBBBBB.....
270 @.....
280 @.....
290 @.....
300 @.....

```

In this way you can get some idea of what your MOB will look like. The entire process of designing a MOB in Simon's Basic takes place on the @ grid.

■■■■■■■■■■

## CMOB

The command CMOB stands for "color MOB" and allows you to define two additional colors that will be used in the designation of a multi-res MOB. The format for the command is very simply

CMOB color 1,color 2

where:

color 1 = 0 - 15

color 2 = 0 - 15

These two parameters are the two additional colors you desire. The primary MOB color is designated in the MOB SET statement just as it is in hi-res MOB design.

When designing a multi-res MOB on an @ grid, the characters B, C, and D are used. Note that the color assignment codes don't work the way you might expect them to in multi-res: the B character now represents the first parameter of the CMOB command, the C character represents the color assigned in the MOB SET command, and the D character represents color 2 in the CMOB command. So a multicolor @ grid might take the following form:

---

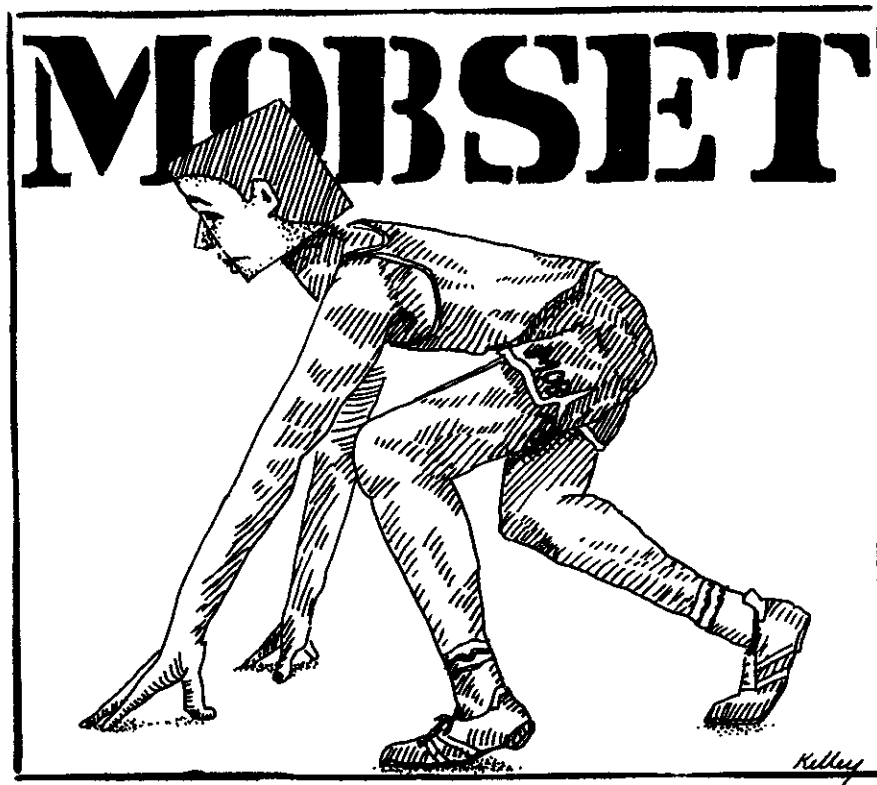
```

1 REM PROGRAM 106
2 REM MULTI-RES MOB EXAMPLEE
3 REM
4 REM-----

```

100 @.....  
 110 @...BBBBBB...  
 120 @...BBBBBB...  
 130 @...BBBBBB...  
 140 @..BBBBBBBB..  
 150 @...CCCCCC...  
 160 @...CCCCCC...  
 170 @....CCCC....  
 180 @.....CC.....  
 190 @...DDDDDD...  
 200 @..DDDDDDDD..  
 210 @..DDDDDDDD..  
 220 @..DDDDDDDD..  
 230 @.C.DDDDDD.C..  
 240 @...DDDDDD...  
 250 @..DDD.DDD..  
 260 @..DDD..DDD..  
 270 @..DDD..DDD..  
 280 @..DDD..DDD..  
 290 @..DDD..DDD..  
 300 @.BBBB..BBBB.





## **MOB SET**

The MOB SET command allows you to set the stage and lift the curtain on a MOB. It performs the initialization process for any designated MOB. The format for the command is

**MOB SET mob priority,memory block,color,screen priority,resolution**

where:

mob priority = 0 — 8

memory block= MOB identifier

color = 0 — 15

screen priority = 0 or 1

resolution = 0 or 1

The parameter priority specifies the number of the MOB you are setting up. This number must be unique for each MOB. The lower the MOB number, the greater its priority over other MOB's. If two or more MOB's are traveling

across the screen, a MOB with a lower number passes over a MOB with a higher number.

The second parameter of the MOB SET command, memory block, defines the memory block from which the MOB shape data will be taken. Use the MOB number of the block to fill this parameter. The next parameter, color, defines the main MOB color. As explained above, the main MOB color is assigned to each mob drawn with a B in the hi-res mode or a C in the multi-res mode.

The next parameter, screen priority, specifies the priority of the MOB over screen data it may encounter. Here you indicate whether you wish the MOB to pass over or under other characters or non-MOB shapes on the screen. A 0 in this position gives the MOB priority over screen data, while a 1 gives screen data priority over MOB.

The last parameter in the MOB SET command, resolution, indicates whether the MOB was created in multicolor or hi-res mode. A 0 in this position indicates hi-res. A 1 defines multi-res.

MOBS IN  
MOTION  
MMOB

The MMOB command means "move" or "manifest MOB." It allows you to display a MOB at one point on the screen and then, if you wish, move it to another location. The format for the command is

**MMOB** mob number,beg x,beg y, end x,end y,expansion,speed

where:

mob number = 0 — 8

beg x = legal x value for MOB size

beg y = legal y value for MOB size

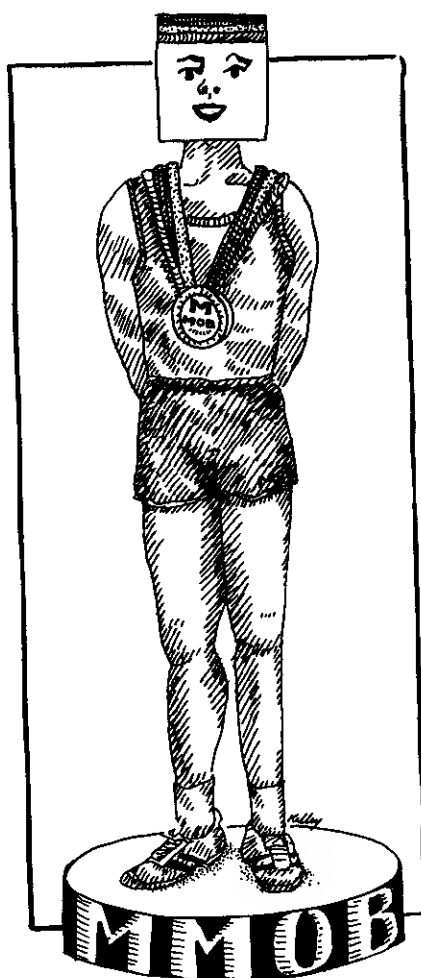
end x = legal x value for MOB size

end y = legal y value for MOB size

expansion = 0 — 3

speed = 1 — 255

The first parameter, mob number, specifies the number of the MOB you wish to display and move. Make sure this number matches the number of the MOB you set in the MOB SET command. The parameters beg x and beg y are the coordinates of the point on the screen where the MOB will be displayed before it is moved. Parameters end x and end y indicate where to finally place the MOB after movement. If you do not wish to move a MOB but just want to display it, use the same values for both start and end screen locations.

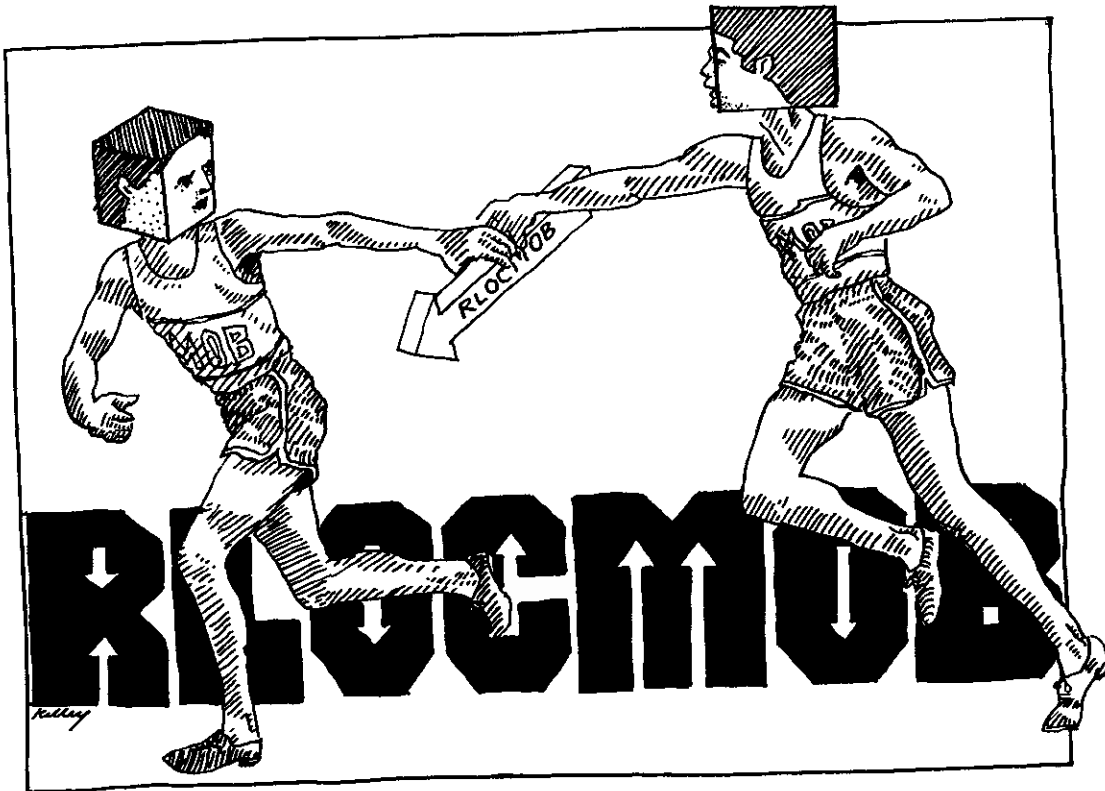


Expansion means the size of the MOB when it is displayed. The expansion numbers and resulting display sizes are shown on the table below:

- 0—MOB is displayed normal size.
- 1—MOB is displayed at double width, normal height.
- 2—MOB is displayed double height, normal width.
- 3—MOB is displayed double width and double height.

The final parameter is speed. This sets the speed of movement of the MOB, and can be a number from 1 to 255. A parameter of 1 is the fastest speed, and 255 is the slowest movement speed attainable.





# RLOCMOB

## RLOCMOB

The command RLOCMOB stands for "relocate MOB," and enables you to move an already displayed MOB to a different location on the screen. Format for the command is much the same as that of MMOB, except the starting position of the MOB is already known:

RLOCMOB mob number, end x, end y, expansion, speed

The parameters end x and end y are the screen coordinates of the point to which the MOB will be moved. All other parameters are the same as encountered in the MMOB command.



■■■■■■■■■■

## MOB OFF

The command MOB OFF does just that—it turns off display of a MOB. The format for the command is

MOB OFF mob number

where:

mob number = 0 — 8

The mob number is the number of the MOB you wish to clear from the screen.

■■■■■■■■■■

## DETECT and CHECK

The commands DETECT and CHECK work together to test for the “collision” of two MOBs or between a MOB and some bit of screen data. There is a bit of a trick to using DETECT and CHECK, but it isn’t too tough to master.

Format for the DETECT command is

DETECT collision type

where:

collision type = 0 or 1

A value of 0 assigned to the collision type parameter causes the C-64 to detect collision between two MOBs. A value of 1 causes the C-64 to detect collision between a MOB and screen data.

The important thing to remember about the DETECT command is that it must be used twice in order to work properly. The first time it is used, it clears the collision register in order that no collision should be detected before it actually occurs. The next time the command is invoked, accurate collision detection will take place.

To effect the detection of an actual collision, the CHECK command is used following the second use of DETECT. The format for the command is

IF CHECK(mob number 1,mob number 2)=0 THEN command

where:

mob number 1 = 0 - 8

mob number 2 = 0 - 8

command = following program statement

The mobs on which you wish to conduct collision tests are the parameters mob number 1 and mob number 2. Follow the THEN with the programming command you want to have take place when collision is detected.

Another format for the command is

IF CHECK(0)=0 THEN command

Use this format to check for collision between any MOB and screen data. Remember to DETECT 1 twice before using this variant.

By now you probably feel buried under tons of commands, few of which make any immediate sense to you. Well, here are some programs to help you get a grip. Play with them, change them, see what makes them work and what makes them stop working. Design your own MOBs to replace the ones used here. Make them collide and detect the collisions. Look at each of the commands, and remind yourself of what each does.

When you get discouraged, take a look at a sprite how-to book working from plain old Basic. Then thank your lucky stars you have Simon's Basic.

The following programs have been designed not for sophistication of effect, but ease of understanding. Play with them until you catch on.

---

```

1 REM PROGRAM 107
2 REM MOVING MOBS--A BEGINNING EXAMPLE
3 REM
4 REM-----
10 DESIGN 0,8192
100 @BBBBBBBBBBBBBBBBBBBBBBBBBBBB
110 @.BBBBBBBBBBBBBBBBBBBBBBBBBB.
120 @..BBBBBBBBBBBBBBBBBBBBBBBB..
130 @...BBBBBBBBBBBBBBBBBBBBBB...
140 @...BBBBBB.....
150 @...BBBBBB.....
160 @...BB.BBB.....
170 @...BB..BB.....
180 @...BB...B.....
190 @...BB...BB.....
200 @...BB...BBB.....
210 @...BB...BBBB.....
220 @...BB...BBBBB.....
230 @...BB...BBBBBB.....
240 @...BB...BBBBBBB.....
250 @...BB...BBBBBBBB.....
260 @...BB...BBBBBBBBB.....
270 @BBBBBBBBBBBBBBBBBBBBBBBBBBBB
280 @BBBBBBBBBBBBBBBBBBBBBBBBBBBB
290 @BBBBBBBBBBBBBBBBBBBBBBBBBBBB
300 @BBBBBBBBBBBBBBBBBBBBBBBBBBBB
310 MOB SET 0,128,2,0,0
320 MMOB 0,0,0,220,70,1,70
330 MMOB 0,220,70,20,90,2,50
340 MMOB 0,20,90,220,190,3,20
350 PAUSE 1:GOTO 320

```

---



---

```

1 REM PROGRAM 108
2 REM FOLLOW THE BOUNCING BALL
3 REM
4 REM-----
10 DESIGN 0,8192
100 @.....
110 @.....
120 @.....
130 @.....
140 @.....BBBBBB.....
150 @.....BBBBBBBBBBBB.....
160 @...BB.BBBBBBBBBB.BB.....
170 @..BBBB.BBBBBBBB.BBBB....
180 @.BBBBBB.BBBBBB.BBBBBB...
190 @.BBBBBBB.BBBB.BBBBBBB...

```

---

```

200 @.BBBBBBBB.BBBB.BBBBBBB...
210 @.BBBBBBBB.BBBB.BBBBBBB...
220 @.BBBBBB.BBBBBB.BBBBBBB...
230 @..BBBB.BBBBBBBB.BBBB....
240 @...BB.BBBBBBBBBB.BB.....
250 @.....BBBBBBBBBBBBBB.....
260 @.....BBBBBB.....
270 @.....
280 @.....
290 @.....
300 @.....
310 MOB SET 0,128,11,0,0
320 MMOB 0,130,50,130,160,0,20
330 MMOB 0,130,160,130,50,0,30
340 GOTO 320

```

---

```

1 REM PROGRAM 109
2 REM MULTI-RES MOB ANIMATION
3 REM
4 REM-----
10 DESIGN 1,8192
100 @.....
110 @...BBBBBB...
120 @...BBBBBB...
130 @...BBBBBB...
140 @.BBBBBBBBB..
150 @...CBCCBC...
160 @...CCCCCC...
170 @....CBBC....
180 @.....CC.....
190 @...DDDDDD...
200 @.CDDDDDDDDDC.
210 @.CDDDDDDDDDC.
220 @.CDDDDDDDDDC.
230 @.C.DDDDDD.C.
240 @CC.DDDDDD.CC
250 @..DDD.DDD...
260 @..DDD..DDD..
270 @..DDD..DDD..
280 @..DDD..DDD..
290 @..DDD..DDD..
300 @.BBBB..BBBB.
310 MOB SET 0,128,10,0,1
320 CMOB 0,6
330 MMOB 0,20,90,250,90,2,110
340 GOTO 330

```

---

---

```

1 REM PROGRAM 110
2 REM MULTIPLE MOB ANIMATION
3 REM
4 REM-----
10 DESIGN 1,8192
100 @.....
110 @...BBBBBB...
120 @...BBBBBB...
130 @...BBBBBB...
140 @..BBBBBBBB..
150 @...CBCCEC...
160 @...CCCCCC...
170 @....CBBC....
180 @.....CC.....
190 @...DDDDDD...
200 @.CDDDDDDDDDC.
210 @.CDDDDDDDDDC.
220 @.CDDDDDDDDDC.
230 @.C.DDDDDD.C.
240 @CC.DDDDDD.CC
250 @..DDD.DDD...
260 @..DDD..DDD..
270 @..DDD..DDD..
280 @..DDD..DDD..
290 @..DDD..DDD..
300 @.BBBB..BBBB.
301 DESIGN 1,8192+64
303 @...BBBBBB...
310 @...BBBBBB...
320 @...BBBBBB...
330 @..BBBBBBBB..
340 @....CCCC....
350 @CC.CBCEC.CC
360 @.C.CCCCCC.C.
370 @.C..CBBC..C.
380 @.C...CC...C.
390 @.C.DDDDDD.C.
400 @.CDDDDDDDDDC.
410 @..DDDDDDDD..
420 @..DDDDDDDD..
430 @...DDDDDD...
440 @...DDDDDD...
450 @..DDD.DDD...
460 @..DDD.DDD...
470 @BDDD....DDDB
480 @BDD.....DDB
490 @BD.....DB
500 @B.....B
520 CMOB 0,6

```

```

525 MOB SET 0,128,10,0,1
530 MMOB 0,170,190,170,190,2,10:PAUSE 2
540 MOB OFF 0
545 MOB SET 1,129,10,0,1
550 MMOB 1,170,190,170,160,2,90
560 MMOB 1,170,160,170,190,2,90
570 MOB OFF 1
580 GOTO 525

```

---

```

1 REM PROGRAM 111
2 REM MORE MOB ANIMATION
3 REM
4 REM-----
5 COLOUR 0,0
10 HIRES 0,0:MULTI 7,0,0
20 CIRCLE 70,90,30,40,1
30 PAINT 71,91,1
90 DESIGN 1,8192+49152
100 @.....C.....
110 @....CCC.....
120 @....CCC.....
130 @....BBB.....
140 @....BBB.....
150 @....BBB.....
160 @...BBBBB....
170 @...BBBBB....
180 @..BBBBB....
190 @..BBBBBBB...
200 @..BBBBBBB...
210 @..BBBBBBB...
220 @..BBBBBBB...
230 @.CCCBBCCC..
240 @CCCCBBBCCCC.
250 @CCC.....CCC.
260 @CC.....CC.
270 @.....
280 @.....
290 @.....
300 @.....
301 DESIGN 1,8192+64+49152
303 @.....C.....
310 @....CCC.....
320 @....CCC.....
330 @....BBB.....
340 @....BBB.....
350 @....BBB.....
360 @...BBBBB....
370 @...BBBBB....

```

```

380 @..BBBBB....
390 @..BBBBBBB...
400 @..BBBBBBB...
410 @..BBBBBBB...
420 @..BBBBBBB...
430 @.CCCBBBCCC..
440 @CCCCBBBCCCC.
450 @CCC..D..CCC.
460 @CC.D.DDD.CC.
470 @...DD..D....
480 @....DDD.....
490 @..D..D..D...
500 @...D..D.....
520 CMOB 6,2
525 MOB SET 0,128,1,0,1
530 MMOB 0,170,230,170,190,2,190
540 MOB OFF 0
545 MOB SET 1,129,1,0,1
550 MMOB 1,170,190,170,10,2,120
570 MOB OFF 1
580 GOTO 525

```

---

```

1 REM PROGRAM 112
2 REM OVER THE BOUNDING MAIN
3 REM
4 REM-----
10 HIRES 3,7
20 PRINT"█":BLOCK█ 0,50,320,210,1
90 DESIGN 0,8192+49152
100 @.....B.B.....
110 @.....BBBBB.BBB.....
120 @....BBBBBBB.BBBB.....
130 @....BBBBBBB.BBBB.....
140 @...BBBBBBB BBBBB.....
150 @...BBBBBBB.BBBB.....
160 @...BBBBBBB.BBBB.....
170 @...BBBBBBB.BBBB.....
180 @...BBBBBBB.BBBB.....
190 @...BBBBBBB.BBBB.B.....
200 @..BBBBBBB.BBBB.BB.....
210 @..BBBBBBB.BBBB.BBB....
220 @..BBBBBBB.BBBB.BBBB...
230 @..BBBBBBB.BBBB.BBBB...
240 @.....B....B.B.....
250 @BBBBBBBBBBBBBBBBBBBBBBB
260 @BBBBBBBBBBBBBBBBBBBBBBB...
270 @.BBBBBBBBBBBBBBBBBBBBB...

```



```
280 @.....  
290 @.....  
300 @.....  
310 MOB SET 0,128,0,0,0  
330 MMOB 0,0,75,319,75,3,200  
340 GOTO 330
```

---





# SOUND FROM SIMON'S BASIC

Simon's Basic isn't as strong at constructing sound as it is in constructing graphics. Still, it is a far cry from the cryptic POKE commands required to get a peep out of your C-64 from plain old Basic. At least in Simon's Basic you have some powerful sound commands at your disposal.

## SIMON'S SOUND COMMANDS

### VOL

Before we begin our programming examples, let's complete an overview of the sound commands, and what they can do for us.

The VOL command stands for volume, and lets us tell the computer how loud to play a note or sound. The format for the command is

VOL volume

where:

volume = 0 — 15

A volume of 0 means no sound at all, while a volume level of 15 is the loudest that it can be played. And until a new VOL level is stipulated, the volume remains where it was set.

### WAVE

It is not easy to explain just what WAVE means without first launching into a lecture on the physics of sound. Let's just say that the C-64 can output basically four different types of sound, and these types of sound correspond to the shapes of certain electronic waveforms.

Using the WAVE command, we tell the Commodore 64 what kind of

“twang” we want our sound to have, and whether we want it to be “noise” as opposed to a pure tone.

The WAVE command includes a lot of compressed information, but you can get the hang of its parameters if you don’t panic. The format for the command is

WAVE voice, control byte

where:

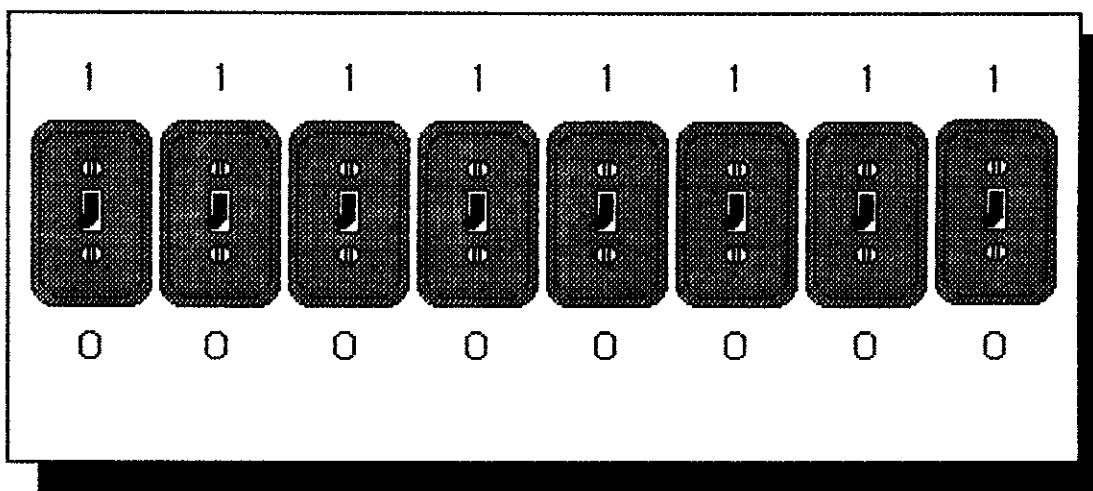
voice = 1 – 3

control byte = 00000000 – 11111111

The parameter voice indicates which of the Commodore’s three voices this WAVE command shall control. The control byte consists of eight single-digit numbers side-by-side, and each must be either a 0 or a 1.

Whenever you encounter mathematics where the only two numbers used are 0 and 1, you’re using a binary numbering system. It is this system which is the crux of your computer’s operation. But there is no theory to learn or mathematics to do here. You may think of the WAVE control byte simply as a bank of eight light switches which must be set in a certain order.

### Binary control of the WAVE command.



The first four switches are the only ones with which we shall concern ourselves. Explanations of the others are beyond the scope of this book. (See your Simon’s Basic documentation for more information on these bit parameters.)

So dealing with the last four switches is easy: for our purposes, we will always set them to 0.

Our concern is with the first four switches. When we choose a waveform, we set just one of these switches, and leave the others off, or in other words set to 0. Let's run through each waveform possibility, and look at the control byte for each of them.

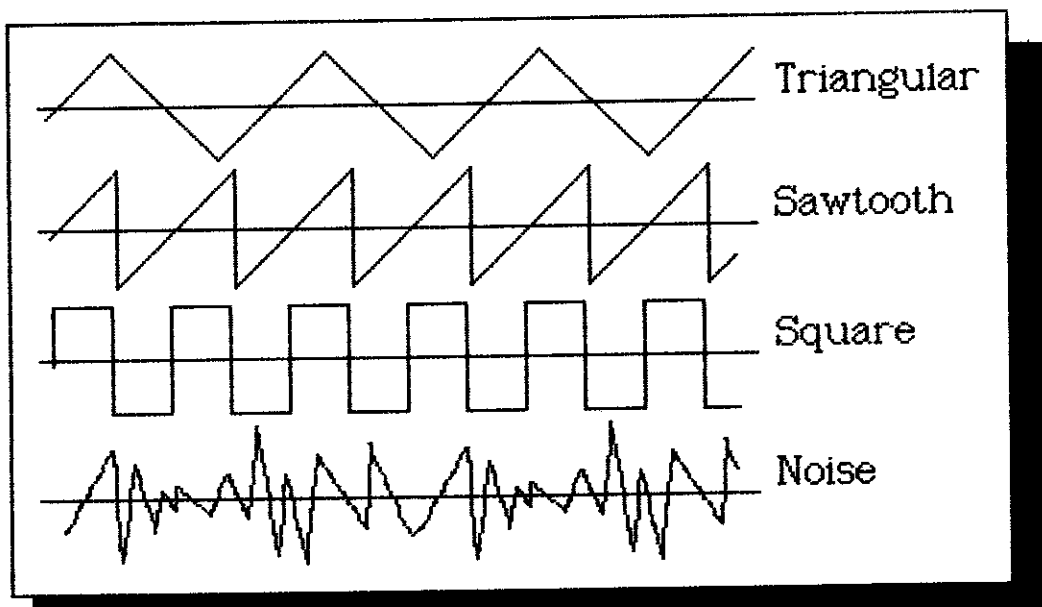
## Noise Waveform

To set up a noise waveform, we set the first switch on while leaving all the rest off. Instead of coming up with any sort of musical tone, this control bit results in "pink noise," which can be used to create sound effects or the percussion voice in music.

The control byte for a noise waveform is as follows:

10000000

C-64 waveform shapes.



## Square Waveform

The next switch setting is for a square wave, which has a sound much like you might expect a computer to sound. It is that flat, pure tone that comes from an L-edged square wave.

To effect a square wave, use the control byte

01000000

■■■■■■■■■■

**Sawtooth  
Waveform**

A sawtooth waveform has a warm, brassy sound. The control byte for a sawtooth wave looks like this:

```
00100000
```

■■■■■■■■■■

**Triangular  
Waveform**

The triangular waveform has a smooth, reed-like, woodwind quality. It sounds like a low flute or recorder.

To call up a triangular waveform from the wave command, we need to set a value of 1 into the fourth switch (or fourth bit) of the control byte. The result is

```
00010000
```

■■■■■■■■■■

**ENVELOPE**

The ENVELOPE command allows us to define the “shape” of the tone we wish to play. It works something like the pedals on a piano.

The format for the command is

```
ENVELOPE voice, attack, decay, sustain, release
```

where:

voice = 1 – 3

attack = 0 – 15

decay = 0 – 15

sustain = 0 – 15

release = 0 – 15

In order to learn about the parameters of the ENVELOPE command, we need to define these terms.

*Voice* indicates which of the Commodore’s three voices we are setting with the ENVELOPE command.

*Attack* determines how quickly from its onset a sound reaches its maximum volume. When a guitar string is plucked, it achieves a maximum volume very quickly. A saxophone might have a much greater *attack* rate.

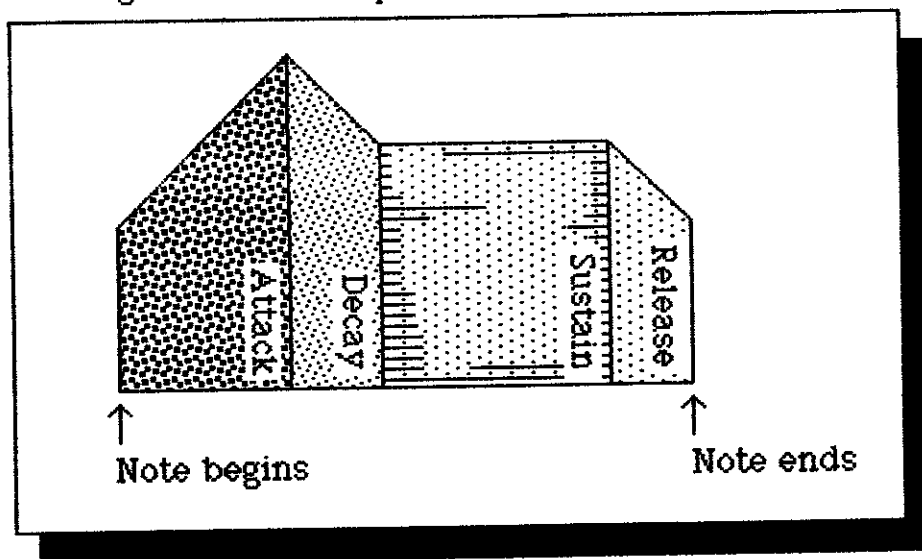
*Decay* indicates the rate at which the volume of a note trails off in volume from its maximum to its midrange. When a piano key is struck, the sound may first sound quite loudly and then trail to a lower midrange, indicating a large decay rate. On a flute, it remains quite constant. This indicates a quick decay.

*Sustain* tells the computer what volume to use following decay—during the mid-range duration of the note.

*Release* determines the rate between the midrange sustain volume and zero volume (meaning the note has finished).

If you were to draw a diagram of the sounding of a note, it might look like this:

Defining a sound envelope.



Certainly you will want to experiment with different rates for each of these parameters. All you need to know is that 0 indicates the quickest rate and that 15 indicates the slowest rate.

## MUSIC

The command MUSIC is the one we use to teach the computer how to play a certain tune or create a certain sound effect. We use it to create a "music string" which is our tune or effect.

The format for the MUSIC command is

MUSIC duration, "command string"

where:

duration = 1 – 255

command string = list consisting of voice code, then beat duration followed by note, for each note

The first parameter in the MUSIC command indicates how long you want one beat of music to be played. Contrary to what you may think, 1 is the longest duration in a MUSIC command, and 255 the shortest.

The next parameter in the MUSIC command is a string of information. It can be represented in the MUSIC command between quotation marks, or an equivalent string. In fact the only way to play long passages of music is to use multiple strings. We will show you some examples up ahead.

The information represented in the command string must take a very certain form. First you must indicate which voice the data pertains to. You do this by making the very first character in the command string an inverse heart (by pressing SHIFT-CLR), then typing the voice to which you want the following data to pertain. Don't worry if you're becoming a bit confused here. The examples up ahead will make things clear to you.

You can follow the inverse heart and voice assignment with the music data itself. This also takes a very specific format. You first indicate the value of each note by pressing a function key. Each function key represents a note value to the MUSIC command. Here is a function/value chart:

### Translating note values to function keys.

sixteenth	■ f1	—————	<b>f1</b>
eighth	■ f3	—————	<b>f3</b>
quarter	■ f5	—————	<b>f5</b>
half	■ f7	—————	<b>f7</b>
full beat	■ f2	—————	<b>SHIFT</b> <b>f1</b>
two beats	■ f4	—————	<b>SHIFT</b> <b>f3</b>
four beats	■ f6	—————	<b>SHIFT</b> <b>f5</b>
eight beats	■ f8	—————	<b>SHIFT</b> <b>f7</b>

After indicating a single note value, you then indicate the note pitch. This is entered in standard letter format (A-G) and is followed by a number from 1

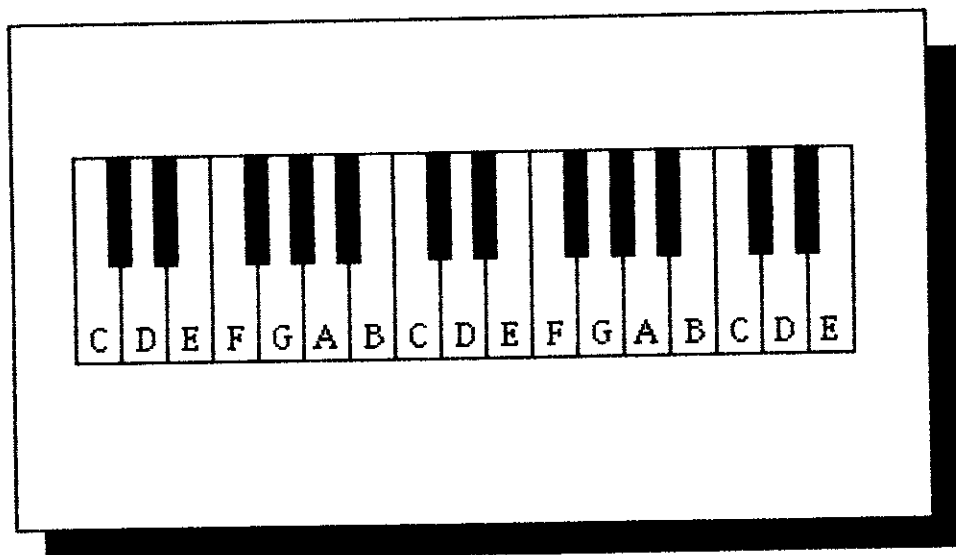


to 8 to indicate which octave the note is in. So a sample command string might look like this

```
10 A$="U12C5E5C5D7F5U"
```

At the conclusion of a MUSIC command string, you should insert another inverse heart, followed by a G. This triggers the release of the last note in the command string.

Piano to C-64 pitch conversion.



## PLAY

The PLAY command does just that. It plays the music you have composed, or the sound effect you have created. It takes the simple format

PLAY mode

where:

mode = 0 – 2

The mode parameter sets the PLAY command one of three possible ways. A 0 in this position turns music off. A 1 plays the music and waits for it

to end before proceeding with the program. A 2 plays the music and continues executing the program, provided you remain in the low-res mode. A parameter of 2 cannot be used in conjunction with hi-res or multi screens.

Confused? Well don't get too frustrated. The examples below are assembled in ascending order of sophistication, and by studying them you should be able to get a grip on sound from Simon's Basic in an hour or so. Experiment with these programs. Use them to plug in values of your own and see what you come up with. But most of all, have fun.

---

```
1 REM PROGRAM 113
2 REM SOUND FROM SIMON'S BASIC
3 REM TRAIN
4 REM-----
10 VOL 8
20 WAVE 1,10000000
30 ENVELOPE 1,2,7,7,2
40 MUSIC 3,"D4"
50 PLAY 1
60 GOTO 50
```

---

---

```
1 REM PROGRAM 114
2 REM SOUND FROM SIMON'S BASIC
3 REM EUROPEAN SIREN
4 REM-----
10 VOL 8
20 WAVE 1,00100000
30 ENVELOPE 1,2,7,7,10
40 MUSIC 4,"C4F4"
50 PLAY 1
60 GOTO 50
```

---

---

```
1 REM PROGRAM 115
2 REM MUSIC FROM SIMON'S BASIC
3 REM A LITTLE BIT OF BACH
4 REM-----
10 A$="E4C4D4E4G4E4F4A4
1G4G4C5B4C5G4E4C4"
15 A$=A$+"D4E4F4G4A4G4E4C
4B3C4D4G3B3D4F4E4D4"
20 VOL 8:WAVE 1,00100000
30 ENVELOPE 1,2,7,7,10
40 MUSIC 2,A$
50 PLAY 1
60 GOTO 50
```

---

---

```

1 REM PROGRAM 116
2 REM MUSIC FROM SIMON'S BASIC
3 REM A LITTLE BIT OF BEATLES
4 REM-----
10 A$="F4G4B4A4F4D4Z4F4G4
1A4C5B4A4B4A4G4A4G4"
15 A$=A$+"1F4G41Z41F41F41G41A41G
41Z4"
20 VOL 8:WAVE 1,00100000
30 ENVELOPE 1,2,7,7,10
40 MUSIC 3,A$
50 PLAY 1
60 GOTO 50

```

---



---

```

1 REM PROGRAM 117
2 REM MULTIPLE MOB ANIMATION
3 REM WITH SOUND EFFECTS
4 REM-----
10 DESIGN 1,8192
100 @.....
110 @...BBBBBB...
120 @...BBBBBB...
130 @...BBBBBB...
140 @...BBBBBBBB..
150 @...CBCCBC...
160 @...CCCCCC...
170 @...CBBC....
180 @...CC.....
190 @...DDDDDD...
200 @.CDDDDDDDDDC.
210 @.CDDDDDDDDDC.
220 @.CDDDDDDDDDC.
230 @.C.DDDDDDD.C.
240 @CC.DDDDDDD.CC
250 @..DDD.DDD...
260 @..DDD..DDD..
270 @..DDD..DDD..
280 @..DDD..DDD..
290 @..DDD..DDD..
300 @.BBBB..BBBB.
301 DESIGN 1,8192+64
303 @...BBBBBB...
310 @...BBBBBB...
320 @...BBBBBB...
330 @...BBBBBBBB..
340 @....CCCC....
350 @CC.CBCCBC.CC
360 @.C.CCCCCC.C.

```

```
370 @.C..CBBC..C.
380 @.C...CC...C.
390 @.C.DDDDDD.C.
400 @.CDDDDDDDDC.
410 @..DDDDDDDD..
420 @..DDDDDDDD..
430 @...DDDDDD...
440 @...DDDDDD...
450 @..DDD.DDD...
460 @..DDD.DDD...
470 @BDDD....DDDB
480 @BDD.....DDB
490 @BD.....DB
500 @B.....B
510 VOL 8:WAVE 1,00100000:ENVELOPE 1,2,7
,7,2
520 CMOB 0,6
525 MOB SET 0,128,10,0,1
530 MMOB 0,170,190,170,190,2,10:PAUSE 2
540 MOB OFF 0
545 MOB SET 1,129,10,0,1
550 MMOB 1,170,190,170,160,2,90
555 MUSIC 1,"1D41E41F41G4":PLAY 1
560 MMOB 1,170,160,170,190,2,90
565 MUSIC 1,"1G21F21E21D2":PLAY 1
570 MOB OFF 1
580 GOTO 525
```

---

# APPENDIX

## Other Sound and Graphics Products for the Commodore 64



There is a lot of hardware and software now available to painlessly introduce the sophisticated graphics and sound potential of the C-64 to the novice user. Although you have learned many programming tricks in this book, using a graphics tablet or a music program can bring even more enjoyment into your relationship with your computer.

Literally dozens of packages on the market today are designed to ease graphics and music production on the Commodore 64. Some are designed as purely software, others as software combined with a special piece of hardware. Cost is generally quite reasonable when compared to the capabilities these products make available.

It would be impossible to provide a list here of all the packages that are currently available—new packages debut almost every day. What we have done here though, is to compile a list of the very best packages to help give you a start in your shopping.

## GRAPHICS PACKAGES

Simon's Basic is in itself a powerful graphics tool—probably the most powerful graphics programming tool around. But using the keyboard to input graphics can be tedious. What is really fun is to draw on the screen as if it were a canvas.

The two tools that are best at doing this on the C-64 are the graphics tablet and the light pen. Each has its own advantages and disadvantages, but each gives you a direct graphics potential you can never achieve from a keyboard. If you want your C-64 to deliver the maximum graphics potential, you should think about purchasing one of the following systems:

### KoalaPad

*Koala Technologies Corporation  
3100 Patrick Henry Drive  
Santa Clara, CA 95050*

The KoalaPad graphics tablet and its accompanying software, KoalaPainter, is an inexpensive way to make your C-64 into an electronic canvas. Using a stylus, you draw on a special tablet and watch your work appear on your television or monitor.

The software provides sophisticated graphics capabilities, including free-hand drawing in all sixteen colors, mirroring functions, zoom magnification, and the design of pictures that can be stored to disk.

### Flexidraw Light Pen

*Inkwell Systems  
7770 Vickers Street  
San Diego, CA 92138*

With a light pen, the process of computerized drawing feels very much like ordinary drawing—you hold a special electronic pen right up to the TV or monitor screen. This can be a little hard on the arm if you don't rest your elbow, but there is no more immediate way to create graphics on your C-64.

The Flexidraw system includes software that makes the Commodore 64 behave almost like an Apple Macintosh. That is, its monochrome graphics capabilities are superlative. After you have created a monochrome picture, you can color it in using another software utility.

### Tech Sketch Light Pen

*Tech Sketch, Inc.  
26 Just Road  
Fairfield, NJ 07006*

The Tech Sketch Light Pen is an inexpensive system with a lot of capability. Its software is nearly identical to that of the KoalaPad, but is of course light-pen driven. The pen itself is not of the same quality as the Flexidraw system, but it is much less expensive and works quite well. The software package, Microillustrator, is among the easiest-to-use packages around. Everything is menu-driven, and you use the pen itself to point to and choose your functions.

.....

## Edumate Light Pen

*Futurehouse, Inc.  
P.O. Box 3470  
Chapel Hill, NC 27514*

The Edumate pen has features quite similar to the Tech Sketch pen's, and though its software is not quite as polished as Microllustrator's, it is very good. What's more, the entire hardware and software system costs only \$60. That is a tough price to beat!

The software for the Edumate system, Peripheral Vision, allows you to choose from thirty-five different fill textures or to design your own. You can copy and move shapes around the screen and place text alongside graphics.

Many other fine graphics packages are on their way. Keep an eye out for the best of them.

.....

## SOUND PACKAGES

For some reason or other, there are not as many high-quality music and sound packages as there are high-quality graphics packages for the C-64. Frankly, most of the music programs released so far leave much to be desired. This is a shame, since the Commodore 64 has incredible music-synthesizing capabilities. Two packages, however, have a lot of capabilities and are not difficult to learn to use.

.....

## Music Construction Set

*Electronic Arts  
2755 Campus Drive  
San Mateo, CA 94403*

Music Construction Set was the first software tool to transform the C-64 into a serious music machine. Using it, you can compose and play back music, and while it's playing back you can watch the notes dance across the screen. Music Construction Set is not only an entertaining program; it can help you learn musical notation and theory as well. Instantaneous playback of your work lets you know where you've made mistakes.

The package makes use of "point-and-click" technology, much like the Macintosh. This makes working with Music Construction Set simple and rewarding.

.....

## MusiCalc

*Waveform Corporation  
1912 Bonita Way  
Berkeley, CA 94704*

MusiCalc is as close to a "dream" piece of sound software as we are likely to see for the Commodore 64 for some time to come. It transforms the computer into a sophisticated music synthesizer.

Using prepackaged software templates, you can get your C-64 to pump out African rhythms or New Wave technobeat. MusiCalc even allows the typewriter keyboard to become a synthesizer keyboard: you can jam along with preset patterns in real time. MusiCalc is easy and fun to use, and shows off the sound capabilities of the Commodore 64 to their full potential.